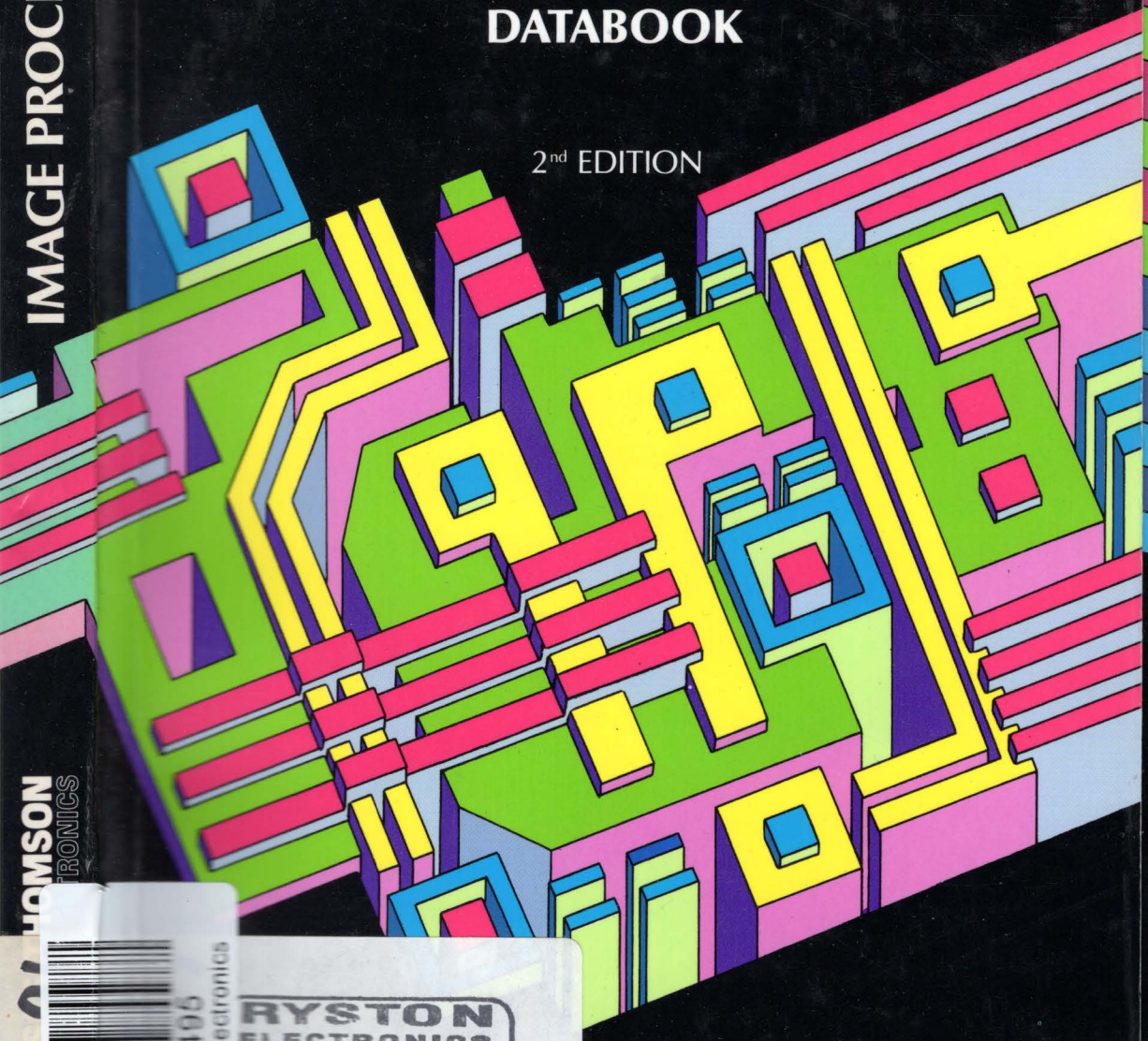
IMAGE PROCESSING

DATABOOK

2nd EDITION



SGS-THOMSON MICROELECTRONICS

IMAGE PROCESSING

SGS-THOMSON MICROELECTRONICS

IMAGE PROCESSING

DATABOOK

2nd EDITION

OCTOBER 1992

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED.

SGS-THOMSON PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS OF LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF SGS-THOMSON Microelectronics.

1. Life support devices or systems are those which (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonably expected to result in significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can reasonably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

TABLE OF CONTENTS

INTRODUCTION

The SGS-THOMSON Image processing databook contains comprehensive data on products for high-speed digital processing and image coding.

High-speed digital products are made of hardwired digital processors. Applications include satellite communication links, studio TV equipments, image processing, radar and sonar.

Image coding products provide video compression and decompression based on the Discrete Cosine Transform algorithm (DCT).

IMAGE CODING
An introduction to the DCT

TV, film and photographs are all common examples of analogue images. As with all analogue signals, analogue images can suffer from degradation when they are processed (Eg. during duplication of a film or video tape or during transmission of a TV signal). Handling images digitally can prevent any degradation in the image and also opens the door to using computers to handle and modify the images.

For example, with conventional colour photography complex chemical, optical and printing processes are required to process, enlarge and duplicate an image. Storing or distributing conventional photographs is also difficult. In contrast a photograph taken using an electronic camera can be stored on a variety of media (Eg. EEPROM, magnetic or optical disc), communicated via standard computer data networks, printed on colour printers or displayed on a monitor and can also be processed or modified by a computer.

The benefits of handling images electronically and digitally are clear. However, the amount of data involved presents a major obstacle. For example, a colour TV picture requires about 200 Mbits/sec of data transmission, and a typical colour photograph requires about 3 Mbytes.

These data rates and storage requirements are much greater than can be supported by today's data transmission networks or stored on current media. Data reduction is the only way to make handling of digital images practical.

Several techniques have been studied to compress images. Some of them are very simple and easy to implement but give little compression. Others are very powerful in terms of compression ratio, but are prohibitive from a cost standpoint.

In a variety of different application areas, transform coding techniques using the Discrete Cosine transform (DCT) are emerging which combine significant compression and impressive image quality with reasonable cost of implementation. Where moving pictures are being handled, transform coding is being augmented by motion compensation to further improve data reduction.

This article describes how DCT/motion - compensated image compression systems work and some of the VLSI products now becoming available to allow their implementation. The key impact of such VLSI techniques will be to allow previously expensive techniques to be used in cost sensitive applications - so, the VLSI implementation of these image compression functions is an enabling technology.

For example videophone and videoconferencing systems have been available for some time. However, as each videophone typically costs $20000 they are little used. New VLSI products will reduce the cost of such systems to levels comparable with other business products (PCs, FAXs, terminals etc.).

Other cost sensitive office and consumer products that become possible are :

- Long playing, CD compatible digital Video-discs.
- Large CD-ROM image databases and interactive CD-ROM video.
- Photographic quality, colour facsimile.
- Photographic quality colour printing and colour desk-top publishing.
- High quality digital video tape recorders (in the future supporting HDTV).

These applications cover several of the most strategic markets (Telecommunications, office/business and consumer).

1 - THE BASIC PRINCIPLES OF THE DCT

In transform coding, the input picture is divided into blocks to take advantage of the localized spatial 2-D correlation of pixels. The block size is determined by the statistical correlation of pixels and by the picture format. Typically, a block size between 8 x 8 pixels or 16 x 16 pixels is used.

The two dimensional transform of a block of pixels results in an uncorrelated coefficient block. The most significant information is concentrated into only a few coefficients which are sufficient to describe the original block. As only these coefficients are required, a first data reduction has been obtained.

INTRODUCTION

One of the most important properties of the DCT is its reversibility; by applying the reverse transform to the coefficient block, the original block of pixels is reconstructed.Compression and decompression are symmetrical. Figure 1 shows an example of this operation. In this example a uniform block is shown where each pixel has the same value (for example blue sky). The result block has only one non-zero coefficient. Thus, the original block of 4 x 4 x 8 bits is coded with only 8 bits.

Obviously, for a natural picture, the process is more complex. The DCT coefficients are related to the spatial frequencies inside the original block of pixels. In the transformed block, the results representing low frequencies (Eg. the average value of the original block) are located in the top left corner, and those representing higher frequencies are in the bottom right corner.

The human eye does not have the same sensitivity to all spatial frequencies. A minimum threshold can be determined for each coefficient in the block. The less significant coefficients are eliminated (without any appreciable loss of quality) in the operation named quantization. Figure 2 shows a block transform of a picture giving the location of the most significant coefficients.

Figure 1 : DCT Operation Example

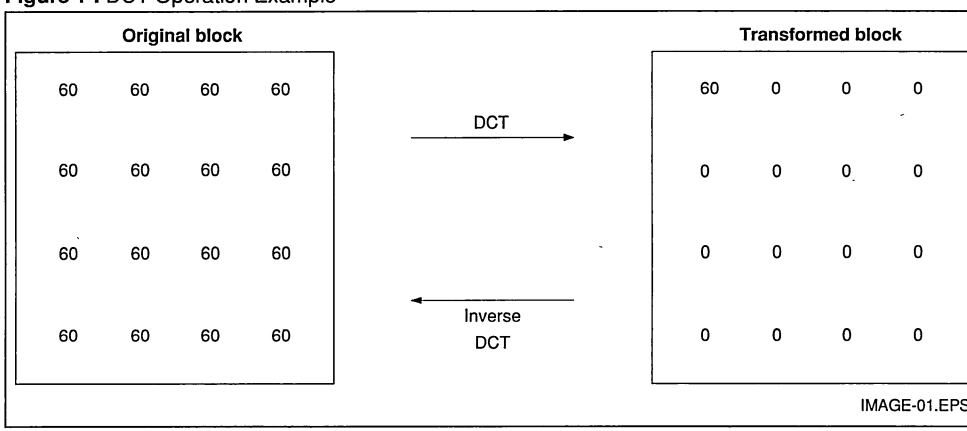
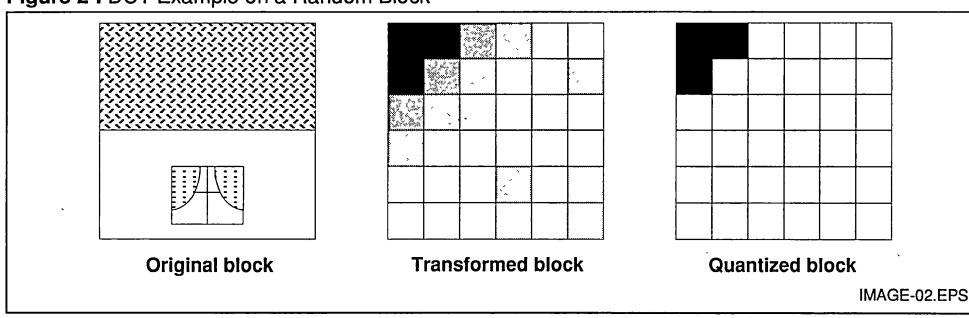


Figure 2 : DCT Example on a Random Block



2 - A STILL PICTURE CODER/DECODER

The block diagram of a still picture coder/decoder based on DCT algorithm is represented in Figure 3.

The still picture coder is divided into four main blocks:

- Block segmentation
- DCT
- Quantization : selection of significant coefficients
- Data packing : Eg. Huffman style coding and CRCs

The core of the system is the DCT operator. To DCT a photograph (1500 x 900 pixels) will require about 40 million multiplications. This might take 20 seconds on a conventional microprocessor or a fraction of a second on a dedicated DCT processor.

The decoder is the inverse of the coder (data depacking, inverse quantization, inverse DCT, scanning conversion as shown in Figure 4). This shows the reversibility of the DCT based algorithm and the symmetry possible in the implementation of a coder and a decoder.

Figure 3 : Block Diagram of a Still Picture Code

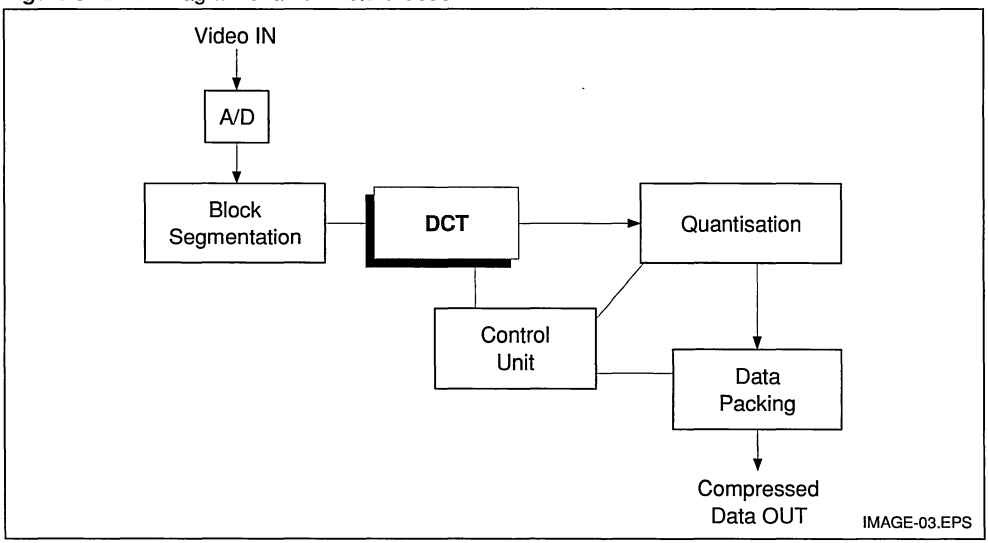
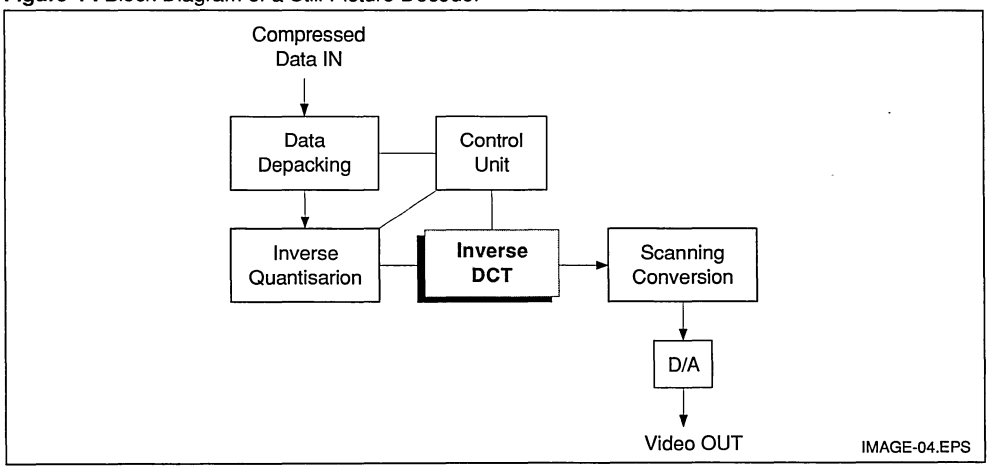


Figure 4 : Block Diagram of a Still Picture Decoder



3 - MOVING PICTURE CODING

A moving picture can be considered as successive still pictures; a TV signal is constituted of 25 or 30 such still pictures per second. The still picture (or intra coding) techniques described previously can be used for moving pictures. Intra coding can provide about a 10:1 data reduction however, many applications require greater data reduction. Processing the difference between successive pictures (inter coding) provides further data compression. In addition motion compensation can improve the compression ratios by an extra factor of 10.

3.1 Motion compensation

Many different methods have been proposed for motion compensation. One of the most practical consists of looking at the position of the current block in the previous picture to determine a motion vector. This is called block matching.

The current block (for which the motion has to be estimated) is projected into the previous picture, and the search for its previous position is done by comparison with all the possible blocks within a search window. It may not be possible to calculate an exact "previous" position so, the best match or "minimum distortion" (defined by the sum of differences between pixels of the two blocks) is used.

Only a limited search window need to be used as the human eye is most sensitive to small movements. Large movements not handled by the motion compensation are seen as differences between successive pictures, and so are handled by the intra-frame DCT coding.

3.2 Moving picture coder

The basic principle of a moving picture coder is to transmit data related to movement in the picture as a set of motion vectors. The motion vectors are only an estimate of the movement within a picture so errors may result between the actual current picture and the prediction derived from the motion vectors. This error can first be concealed, or made less objectionable, by a simple low pass "loop filter" (see Figure 6). The remaining error between the actual and predicted images is DCT coded and transmitted with the motion vectors. Once the "base" image has been transmitted (by intra coding) the only data required is the motion vectors describing movement in the picture and DCT coded "error" information.

Depending on the compression ratios and image quality desired, different mixtures of single picture (intra-frame) coding and motion compensation inter-frame coding can be used. Intra-frame coding also allows correction of any cumulative errors that may build up and in applications like video disc players allows random access into a video sequence. Figure 6 shows a block diagram of a moving picture coder.

Figure 5 : Block Matching Algorithm

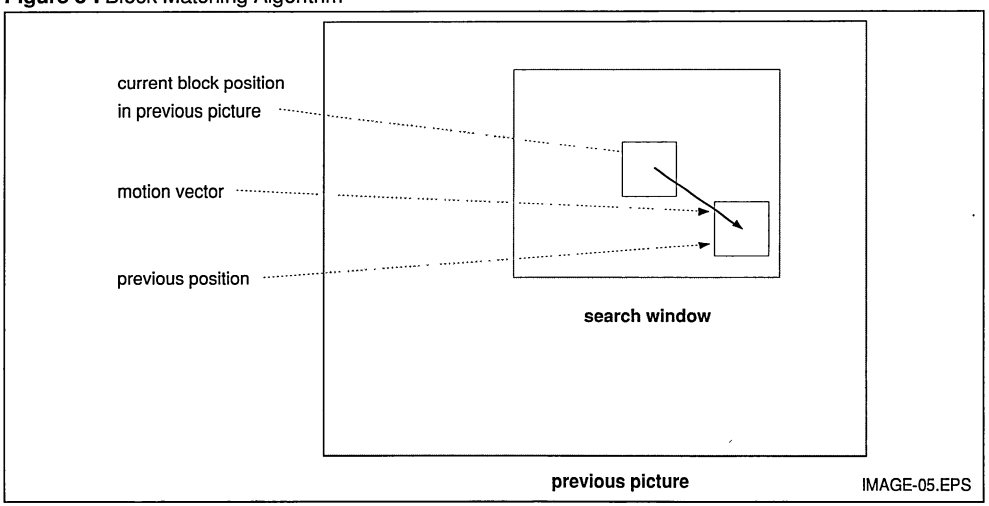
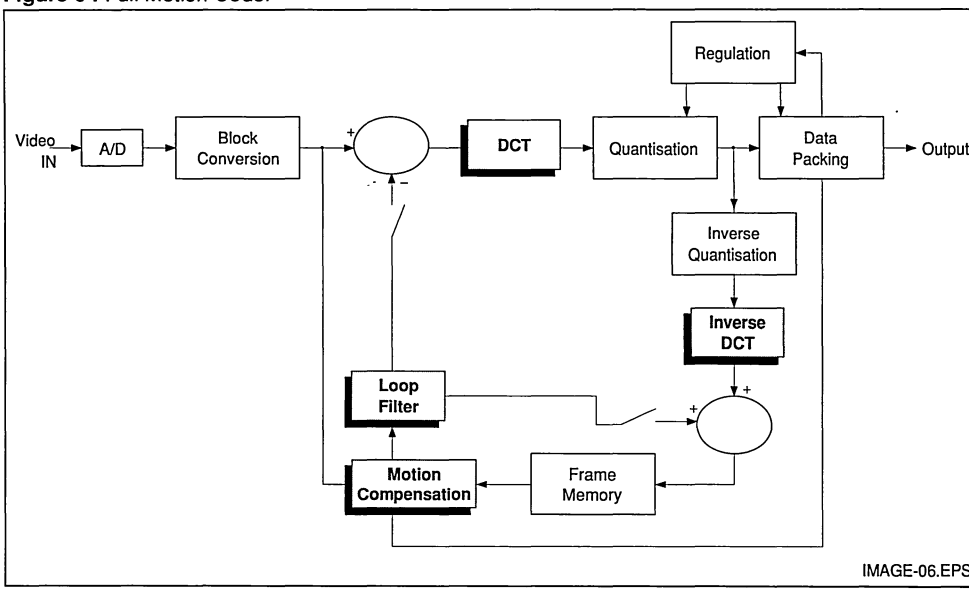


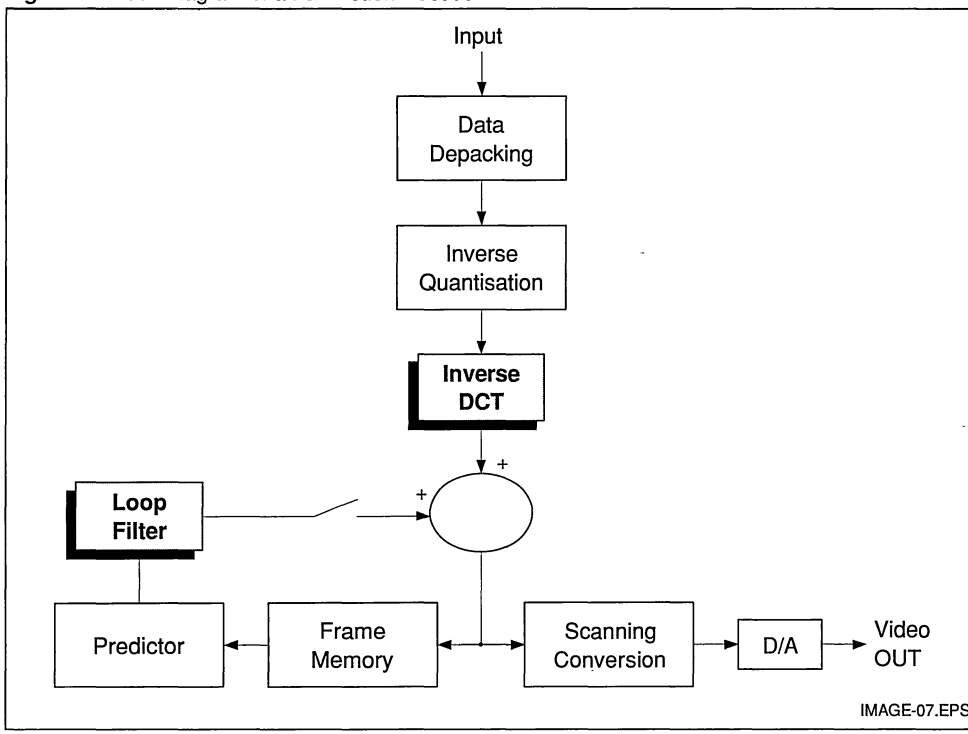
Figure 6 : Full Motion Coder



3.3 Moving picture decoder

The moving picture decoder is much simpler. Compressed picture or picture-error data (from intra or inter-frame coding) is recovered by the inverse cosine transform. This data is either used directly as picture information (intra-frame coding) or as error information correcting the values predicted with the motion vector information. An advantage of this method is that in the decoder, except for the inverse DCT, there is no huge computation requirement. The main computation has been done in the coder.

Figure 7 : Block Diagram of a Full Motion Decoder



4 - SUMMARY ON DCT AND MOTION COMPENSATION ALGORITHMS

International organizations such as the ISO and CCITT are looking at standards for image coding using DCT algorithm. This is a recognition of the advantages of this algorithm which can be summarized as follows :

- DCT provides an efficient coding scheme regarding compression ratios and image quality.
- DCT algorithm is reversible and symmetrical, this allows real time coding and decoding on similar hardware.

ALPHANUMERICAL INDEX

SELECTION GUIDE

DIGITAL SIGNAL PROCESSING DEVICES

Part Number	Function	DataRate (MHz)	MOPS	Package
IMSA100-21	1 Dimensional filter / convolver, 32 taps	21	80-320	PGA84

IMAGE CODING DEVICES

Part Number	Function	Data Rate (MHz)	Package
IMSA121	Discrete Cosine Transform Processor (DCT) 8 x 8 pixel block size operation DCT, IDCT, Filter, Transpose operation Post-adder, pre-substractor	20	PLCC44
STV3200	Discrete Cosine Transform Processor (DCT) Multi pixel block size operation from 4 x 4 to 16 x 16 pixels	15.0	DIP40 PLCC44
STV3208	Discrete Cosine Transform Processor (DCT) 8 x 8 pixel block size operation Zig-Zag scan of coefficients	20/27	DIP40 PQFP44
STI3220	Motion Estimator Processor Block matching, full search algorithm -8/+7 displacements 8 x 8 to 16 x 16 pixel block size operation	18	PQFP144

IMAGE PRE-POST PROCESSING DEVICES

Part Number	Function	Data Rate (MHz)	Package
IMSA110	2 dimensional filter / convolver 21 x 1 or 7 x 3 kernel 3 delay lines back-end processor	20	PGA100
STV8438	Triple 8-bit D/A converter voltage outputs internal voltage reference External analog inputs with switching capability	70	SDIP42 PQFP44

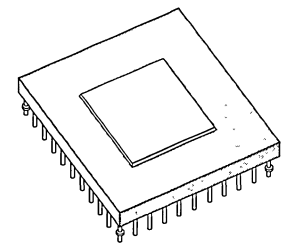
DATASHEETS

DIGITAL SIGNAL PROCESSING DEVICES

IMSA100

CASCADABLE SIGNAL PROCESSOR

- FULL 16 BIT, 32 STAGE, TRANSVERSAL FILTER
- FULLY CASCADABLE WITH NO SPEED DEGRADATION OR REDUCTION IN DYNAMIC RANGE
- COEFFICIENTS SELECTABLE AS 4, 8, 12, OR 16 BITS WIDE
- DATA THROUGHPUT TO 15.0 MHZ
- HIGH SPEED MICROPROCESSOR COMPATIBLE INTERFACE
- DATA INPUT AND OUTPUT THROUGH DEDICATED PORTS OR VIA THE MICROPROCESSOR INTERFACE
- FULLY STATIC HIGH SPEED CMOS IMPLEMENTATION
- SINGLE +5V ± 5% OR ± 10% POWER SUPPLY VARIANTS
- TTL AND CMOS COMPATIBILITY
- LESS THAN 2W POWER DISSIPATION
- STANDARD 84-PIN PGA



PGA84
(Grid Array Package)

APPLICATIONS

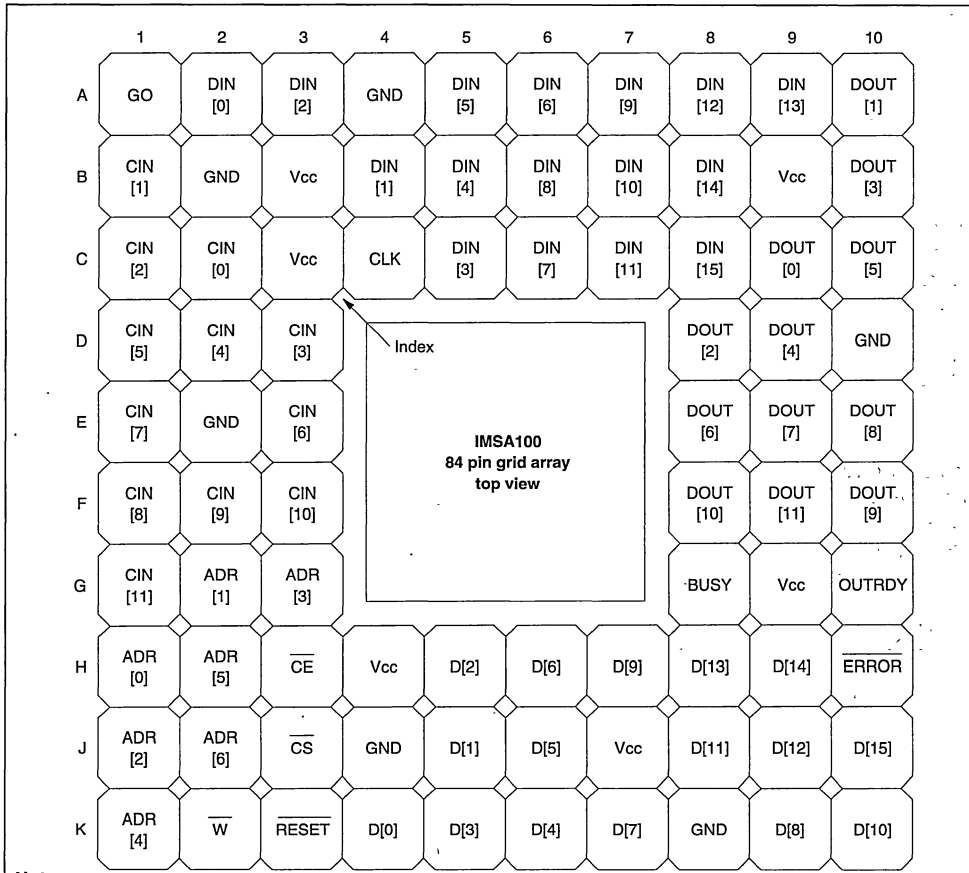
- Digital FIR filtering
- High speed adaptive filtering
- Correlation and Convolution
- Discrete Fourier Transform
- Speech processing using Linear Predictive Coding
- Image processing
- Waveform synthesis
- Adaptive and fixed equalizers and echo cancellers
- Spread spectrum communication
- Beamforming and beamscanning in sonar and radar
- Pulse compression
- High speed fixed point matrix multiplication

ORDERING INFORMATION

Part Number	Package	Clock Speed	Temperature
IMSA100-G21I	Ceramic Pin Grid Array	21MHz	-40°C, +85°C
IMSA100-G21S	Ceramic Pin Grid Array	21MHz	0°C, +70°C

A100-01.TBL

PIN CONNECTIONS



Note
All **V_{CC}** pins **must** be connected to the 5 Volt power supply.
All **GND** pins **must** be connected to ground.

A100-01.EPS

1. INTRODUCTION

The IMSA100 is a high speed, high accuracy 32 stage transversal filter. Its flexible architecture allows it to be used as a 'building block' in a wide range of Digital Signal Processing (DSP) applications. The part is capable of performing high speed DFTs, convolution and correlation, as well as many filtering functions.

The input data word length is 16 bits, and coefficients are programmable to be 4, 8, 12 or 16 bits wide; two's complement numerical formats are used for both data and coefficients. The coefficients can be updated asynchronously to the system clock during normal operation, allowing the chip to be used in a variety of adaptive systems. The IMSA100 can also be cascaded to construct longer transversal filters with no additional logic or degradation in speed, whilst preserving a high degree of accuracy. The device is controlled through a standard memory interface, allowing use with any general purpose microprocessor. Data communications can be either through the memory interface, or through dedicated data ports.

2. DESCRIPTION

The IMSA100 is a 32 stage, cascadable, digital transversal filter. The general canonical transversal filter is shown in Figure 1. An alternative, and functionally equivalent filter is shown in Figure 2. It is this second realisation that is used in the IMSA100, where the input signal is supplied in parallel to all 32 multipliers, and the delay and summation operations are performed in a distributed manner.

Each data sample loaded into the IMSA100 is fed in parallel to all 32 stages. At each stage the current input sample is multiplied by a coefficient stored in memory, and added to the output of the previous stage delayed by one clock cycle. The filter output at time t=kT is given by:

$$y(kT) = C(0) \times x(kT) + C(1) \times x((k\text{-}1)T) + \ldots$$
$$\ldots + C(N\text{-}1) \times ((k\text{-}N\text{+}1)T)$$

where $x(kT)$ represents the kth input data sample, and $C(0)$ to $C(N\text{-}1)$ are the coefficients for the N stages.

While the IMSA100 architecture is designed as a transversal filter it contains many features which allow it to be used in a wide range of signal processing applications, e.g. adaptive filtering, matrix multiplication, discrete Fourier transforms, correlation and convolution. Figure 3 shows the users view of the IMSA100.

The IMSA100 has four interfaces through which data can be transferred. The memory interface port allows access to the coefficent registers, the configuration and status registers and the data input and output registers for the multiplier accumulator array. Three dedicated ports are also provided, allowing high speed data input and output to the IMSA100 and the cascading of several devices.

Typically a microprocessor will configure the IMSA100 via the memory interface, then in a simple system data input and output can be performed through the data input (DIR) and data output (DOL, DOH) registers. Alternatively in a higher performance system data transfer may be performed via the dedicated input and output ports. A typical IMSA100 based system is shown in Figure 4. Simple high-throughput fixed-configuration systems can be implemented by clocking the configuration information into the IMSA100 from a ROM.

Figure 1 : Canonical Transversal Filter Architecture

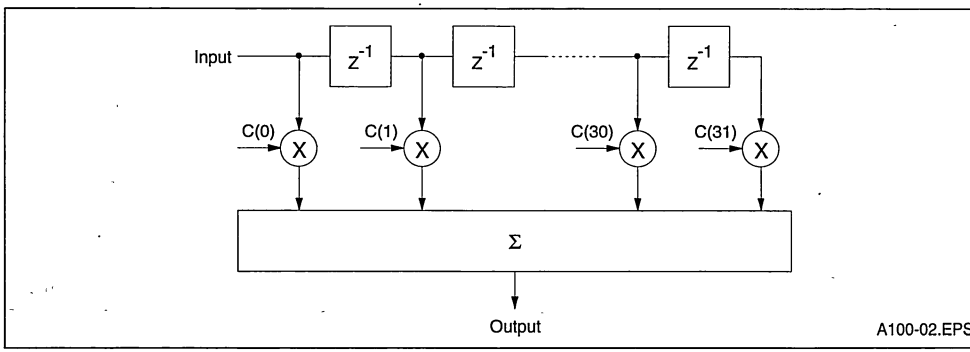


Figure 2 : Modified Transversal Filter Architecture

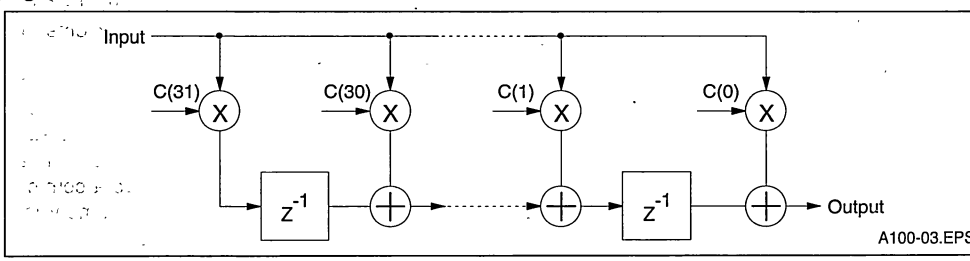


Figure 3 : IMSA100 User Model

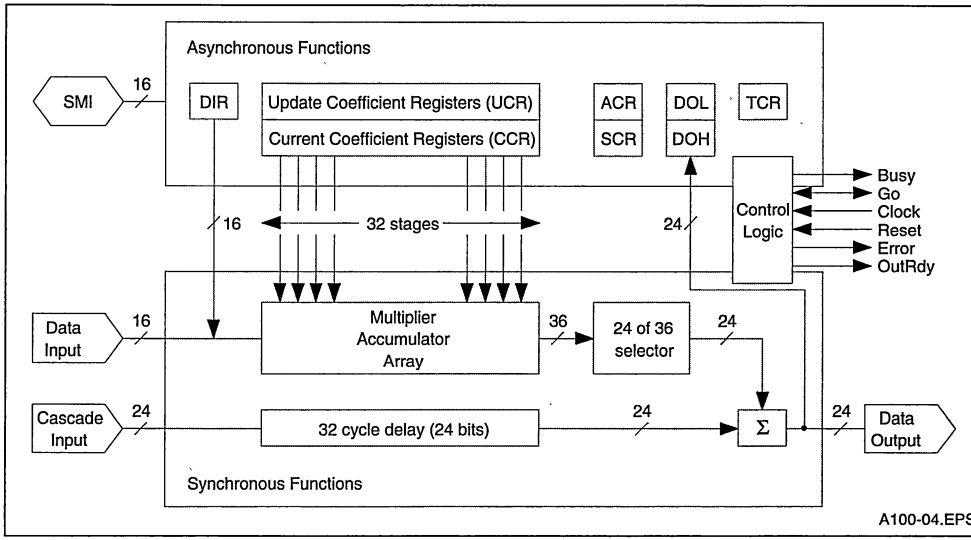
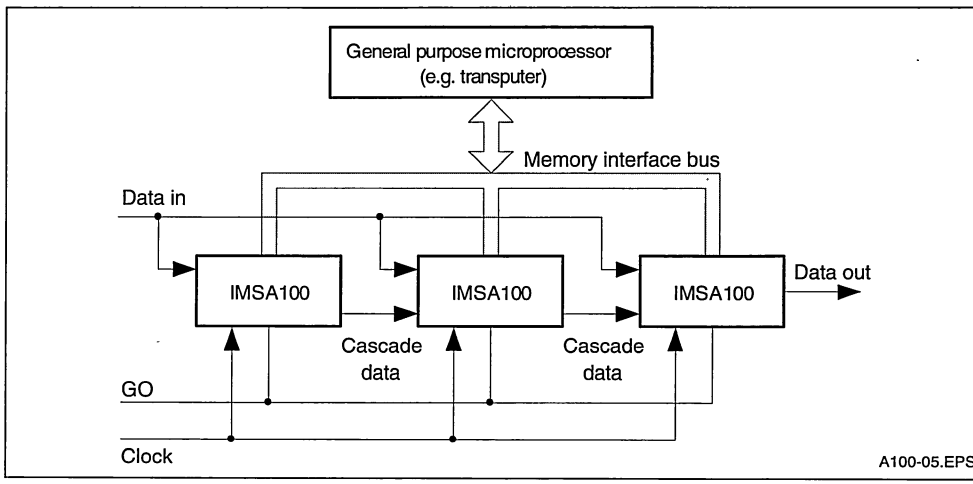


Figure 4 : Simple IMS100 Based System



The IMSA100 input data word width is 16 bits. The coefficient words can be programmed to be 4, 8, 12, or 16 bits wide. There is a trade off between the coefficient size and the speed of operation. If the coefficient word is L_C bits wide and the clock frequency applied to the IMSA100 is F then the maximum data throughput is $\frac{2xF}{L_C}$ So, for an IMSA100 operating from a 20.8MHz clock and using 4-bit coefficients the maximum data throughput is 10.4MHz, similarly for 16-bit coefficients the throughput is 2.6MHz.

To preserve complete numerical accuracy, no truncation or rounding is performed on the partial products in the multiplier accumulator array. The output of this array is calculated to full precision (36 bits). A programmable barrel shifter is located at the output of this array, which allows one of five 24 bit

fields to be selected from the 36 bit result. The selected 24 bits are always correctly rounded and are sign extended before being output. The selection required can be determined from analysis of the coefficients and input data used in a given application.

Two banks of coefficients are provided. At any instant one set of coefficients is in use within the multiplier accumulator array, the other set being accessible via the memory interface. Once a new set of coefficients has been loaded, the two coefficient banks can be interchanged by performing a write operation to the 'Bank Swap' bit of a control register.

So that devices can be cascaded (eg. to construct longer transversal filters), a 32 stage, 24 bit wide, shift register and 24 bit adder is included on chip. The output of one chip is connected directly to the cascade input of the next. The output of the shift register is added internally to the output of the programmable barrel shifter to give the final 24 bit output from the chip. To minimise pin count and external buses, the data output and the cascade input ports transfer 24 bit words as a pair of 12 bit words across a 12 bit wide multiplexed interface.

As IMSA100s can be cascaded there is a price / performance trade off for most IMSA100 systems. For example, a correlation application could achieve high performance by using a cascade of IMSA100s sufficiently long to hold one of the waveforms being correlated in its coefficient registers and sending the other waveform involved in the correlation along the cascade of IMSA100s. A cheaper and slower solution would be to use a smaller number of IMSA100s and to decompose the single long correlation into a sequence of shorter correlations, the results of which are then summed.

3. PIN DESIGNATIONS

System services

Pin	In/out	Function
V_{CC}, GND		Power supply and return
CLK	in	Input clock
$\overline{\text{RESET}}$	in	System reset
$\overline{\text{ERROR}}$	out	Numerical overflow error
BUSY	out	Bank swap in progress

Synchronous input/output

Pin	In/out	Function
GO	in/out	Initiate input/computation/ output cycle
DIN[0-15]	in	Data input port
DOUT[0-11]	out	Data output port
CIN[0-11]	in	Cascade input port
OUTRDY	out	Output data ready

Asynchronous input/output

Pin	In/out	Function
D[0-15]	in/out	Memory interface data bus
ADR[0-6]	in	Memory interface address bus
$\overline{\text{CS}}$	in	Memory interface select
$\overline{\text{CE}}$	in	Memory interface enable
$\overline{\text{W}}$	in	Memory interface write enable

Notes
Signal names are shown with an overbar if they are active low, otherwise they are active high.

3.1 System services

System services include all the necessary logic to start up and maintain the IMSA100.

Power

Power is supplied to the device via the V_{CC} and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100nF low inductance (e.g. ceramic) capacitor between V_{CC} and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to V_{CC} and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

CLK

The clock input signal **CLK** controls the timing of input and output on the three dedicated ports and controls the progress of data through the multiplier accumulator array.

$\overline{\text{RESET}}$

When the IMSA100 is reset the control logic within the IMS A100 will be reset and the ACR, SCR and TCR will be initialised to their default values.

Note that neither the internal data path registers nor the coefficient registers are affected by the reset. Resetting the device initialises the SCR to its default setting.

So, depending on the setting of SCR before a reset, a reset may also be a device reconfiguration. The sequence of operations required to return the device to a defined state following reconfiguration is described under SCR in the register description.

A reset is initiated automatically when power is first applied to the device.

This reset will be completed once four cycles of **CLK** have occured after V_{CC} is valid. Alternatively reset can be initiated by taking $\overline{\text{RESET}}$ low. This reset will be completed after at least two cycles of **CLK** have occured while $\overline{\text{RESET}}$ is held low. $\overline{\text{RESET}}$ should be held low for at least 200ns. Normal device operation can then continue after $\overline{\text{RESET}}$ is taken high.

The reset should be completed before either the synchronous or asynchronous parts of the device are used.

$\overline{\text{ERROR}}$

If asserted, this pin indicates an error condition has occured, and that the condition has not been cleared. The error condition results from a numerical overflow in either the final adder or in the field selector. To allow this signal to be wire ORed between all the devices in a cascade and hence to be used as an interrupt signal to the host processor, the $\overline{\text{ERROR}}$ outputs are open collector.

If suitably armed before the error occured the ACR error bits can be read to discriminate the two error sources. The error bits in the ACR and the error condition can be cleared and then the error bits armed to detect further errors by writing values to the ACR. The sequence of values that should be written to the ACR error bits is 0 followed by 1. An error condition can only be cleared if the error bits were suitably armed before the most recent error occured.

The ACR error bits may not observe an error occuring between clearing and arming the error bits. So, when clearing an error and arming the error bits precautions should be taken to ensure that no new error occurs. For example, first prevent the IMSA100 from initiating computation on new data; second wait for any results pending to be output; then clear and rearm. The ACR error bits will observe any error occuring after they are armed. Thus, if an error occured before the ACR error bits were armed it may be necessary to arm the error and then force an error before proceeding to clear the error (as described above).

Following power up the contents of the multiplier accumulator array and cascade path are indeterminate. As this indeterminate data flushes through a system of one or more IMSA100s errors are likely to occur. Similarly, altering the device configuration defined by the SCR is likely to result in errors. The sequence of operations required to return the device to a defined state following reconfiguration is described under SCR in the register description section of this specification.

BUSY

When high this pin indicates that an exchange of data between the Current and Update Coefficient Registers is in progress. Under certain conditions the duration of **BUSY** may be vanishingly small. **BUSY** will be active if the bank swap is caused by setting ACR[0] to request a single bank swap or when SCR[2] is set selecting Continuous Swap mode. The detailed behaviour is described in the bankswap timing diagrams.

3.2 Synchronous input/output

GO

The **GO** signal initiates a cycle of data input, computation and output. An IMSA100 configured as a slave will monitor the **GO** signal on the rising edge of **CLK** one cycle before it is ready to accept more data and on every rising edge thereafter until **GO** is found to be high.If **GO** is high then data input will occur on the next rising edge of **CLK**. If **GO** is low when it is sampled no new data input will occur.

In a cascade of IMSA100s one IMSA100 may be configured as a master. The master IMSA100 will drive its **GO** pin high after data has been written into its Data Input Register indicating that new data is available and that the slave IMSA100s in the casacade should start an input, computation, output cycle. When the **GO** signal goes low new data can be written to the IMSA100s. Typically a host processor will write simultaneously to the Data Input Registers of all the IMSA100s in the cascade. The host will then monitor the **GO** signal before writing new data to the cascade.

DIN[0-15]

This 16 bit wide data input port allows high speed data input to the IMSA100. The timing of this input is controlled by the **CLK** and **GO** signals. In a cascade of IMSA100s the 16 bit wide input data path and the **CLK** and **GO** signals will be bussed to all devices.

DOUT[0-11]

This 12 bit data port outputs the result from the IMSA100. The 24 bit result is multiplexed through this port as two 12 bit words, the least significant word being output first. The most significant word is output second and remains on the data pins until a new data output sequence is about to start. The **OUTRDY** signal can be used to latch these words into external circuitry. In a cascade of IMSA100s the **DOUT** pins of one device connect to the **CIN** pins of the next device in the cascade.

CIN[0-11]

The Cascade Input allows multiple IMSA100s to be cascaded. A 24 bit word is input as two 12 bit words the least significant word being input first. The 24 bit word is delayed by a shift register and summed with the output of the multiplier accumulator array. The delay from a word being input on the cascade input to that word affecting the data output is 32 data input cycles. In a typical IMSA100 based system the cascade input of each device will be connected to the data output **DOUT[0-11]** of the previous IMS A100 in the cascade. The Cascade Input of the first device in the cascade will normally be connected to ground.

OUTRDY

The output ready signal **OUTRDY** goes low just after the least significant data output word is available on the **DOUT** pins and goes high just after the most significant word is available. The rising edge of **OUTRDY** also indicates that the Data Output registers (DOL, DOH) contain the new result word. Thus the **OUTRDY** signal can either be used to latch the output of the IMSA100 into external logic or to indicate that output of the IMSA100 can be read through the memory interface from the Data Output registers.

3.3 Asynchronous input/output

$\overline{CS}$

This pin selects the chip; if chip select $\overline{CS}$ is low an access to the memory interface will be enabled. This signal is usually asserted by the host processor's address decoder at the beginning of a memory cycle.

$\overline{CE}$

The chip enable pin. The memory interface on the IMSA100 appears to the system controlling it as 128 words of static RAM. The chip enable $\overline{CE}$ signal is similar in operation to the chip enable signal found on static RAMs. When $\overline{CE}$ is high the chip select, write enable and the address inputs are ignored and the memory interface data bus is tri-state. When chip enable is low a single read or write access is made to one of the registers within the IMSA100. Accesses to the memory interface can occur completely asynchronously to operations on the data in, cascade in and data output ports **DIN[0-15]**, **CIN[0-11]** and **DOUT[0-11]**.

$\overline{W}$

The write enable pin indicates whether the access to the IMS A100 memory interface is to be a write or a read. If $\overline{W}$ is low a write access is indicated.

ADR[0-6]

The seven bit address bus comprises pin **ADR[0-6]**. The seven bit binary value applied to the address inputs of the IMSA100 indicates which register is to be accessed.

D[0-15]

During a write to the memory interface a 16 bit word is applied to data bus pins **D[0-15]**. This word will be latched on the rising edge of chip enable $\overline{CE}$ at the end of the cycle. During a read cycle the contents of the location accessed are placed on the data pins. When $\overline{CE}$ is high the data signals are tri-state.

4. REGISTER DESCRIPTION

The memory map shown below indicates the primary addresses for each register. All locations between decimal addresses 64 and 75 inclusive are uniquely decoded. This group of registers is shadowed at other locations up to the 128 word boundary. The effect of reading and writing to areas in the memory map other than those shown in the table is undefined.

If the user wishes to initialise the device from a ROM addressed by a clocked counter, one of the following options applies:

1 Restrict the counter to count only from 0 to 68; this avoids writing to the data registers as well as the shadow locations.

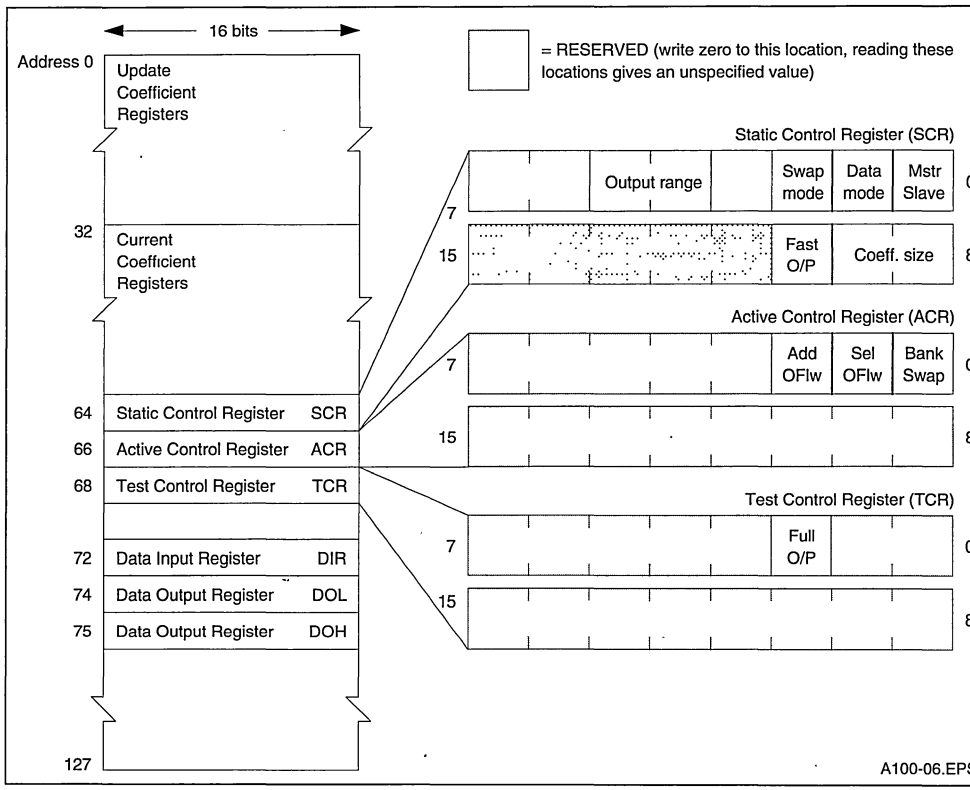
2 Count down from 127 to zero. The initialization at the lower addresses will override spurious ones at the higher shadowed addresses.

4.1 Memory Map*

Register	Address decimal	Address hex	Function
CCR[0-31]	32-63	20-3F	Current Coefficient Registers
UCR[0-31]	0-31	00-1F	Update Coefficient Registers
SCR	64 65	40 41	Static Control Register Unused location
ACR	66 67	42 43	Active Control Register Unused location
TCR	68	44	Test Control Register
DIR	72	48	Data Input Register
DOL	74	4A	Data Output Register (Least Significant Word)
DOH	75	4B	Data Output Register (Most Significant Word)

* All other locations accessible via the memory interface of the IMSA100 are reserved.

Figure 5 : IMSA100 Memory Map



4.2 Registers

CCR[0–31]

The Current Coefficient Registers contain the coefficients currently being used by the multiplier accumlator array. CCR[0] (decimal address 32) corresponds to the coefficient register of the multiplier accumlator nearest the output of the IMSA100; i.e. this location is equivalent to C(0) in Figure 2.

Similarly CCR[31] (decimal address 63) corresponds to C(31). The Current Coefficient Registers can be read from at any time and can be written to provided that no data processing is taking place. The effect of writing to the Current Coefficient Registers while data is being processed is undefined.

UCR[0–31]

The Update Coefficient Registers are equivalent to the Current Coefficient Registers, with the exception that the values in the Update Coefficient Registers are not currently in use within the multiplier accumlator array and can therefore be written to at any time.

A bank swap operation is equivalent to an exchange of data between the Update Coefficient Registers and the Current Coefficient Registers.

SCR

The Static Control Register contains the control bits which configure the IMSA100 and are unlikely to need updating after their initial configuration. The contents of the Static Control Register are not affected by the IMSA100 and can be read at any time.

Reconfiguring the SCR may result in indeterminate data values within the IMSA100 system. These values may in turn result in errors. After reconfiguring the SCR the following sequence should be followed to return the IMSA100 system to a defined, error free condition:

1 Arm error bits in ACR.
2 After SCR has been reconfigured **GO** should be held low for 20 cycles of **CLK**.
3 A series of suitable data values should then be flushed through the IMSA100 system.
4 Any errors generated should then be cleared.
5 The IMSA100 system is then ready to commence normal operation.

ACR

The Active Control Register contains status and control bits which are likely to be accessed during normal operation of the IMS A100; i.e. when handling error conditions and when requesting single coefficient bank swaps.

TCR

The Test Control Register is used for test purposes. One of the test modes provides access to the least significant part of the multiplier accumulator array output.

DIR

The Data Input Register. The IMSA100 can be configured to either take its input data from the **DIN** pins or from the Data Input Register. If the IMSA100 is configured as the master of a cascade of IMSA100s the **GO** signal will be driven in response to writing data into the Data Input Register.

In a small IMSA100 based system the Data Input Registers of all the devices in the cascade will normally be mapped into the same location within the address space of the processor controlling the cascade. Thus a single write operation can write data to all devices, the master IMSA100 generating the **GO** signal for the slaves. The Data Input Register is write only.

DOL

The least significant word of the Data Output Register. The output data from the IMSA100 is available from both the **DOUT[0–11]** pins and from the Data Output Registers. The value held in the Data Output Registers is the 24 bit output word, sign extended to 32 bits. DOL contains the least significant 16 bits of the 24 bit result; the register is read only.

DOH

The most significant word of the Data Output Register. The DOH register contains the most significant 8 bits of the 24 bit output word generated by the IMSA100. The most significant 8 bits of DOH are the sign extension of the output word. DOH is read only.

The remainder of this section describes the register details bit by bit. Each section commences with the name of the register with the bit number(s) followed by the default value, in the general format:

- **Name**

REGISTER[MSB–LSB]	Default: MSB LSB

The least significant bit of a register is bit 0.

* in the tables indicates the default state of the register bit(s).

4.3 Static control register

- **Fast Output**

SCR[10]	Default: 0

The Fast Output bit controls the way in which the 24 bit output of the IMSA100 is multiplexed across the 12 bit wide **DOUT** port. The interval between data output cycles is the same for both Normal and Fast output modes.

The difference between the modes is the time division between the least and most significant words. In fast output mode the least significant 12 bit word is available for the minimum period possible, thus allowing the most significant word to be output at the earliest possible instant. In normal output mode the least significant word is available for the same length of time as the most significant word (unless the duration of the most significant word is extended by idle cycles).

The timing constraints on data output in Normal mode are significantly simpler than those in Fast mode. Fast mode should be considered a special mode which is only used where the early availability of the output words is important, e.g. an adaptive system where the filter coefficients are being modified in response to the output data.

All devices in a cascade of IMSA100s should be configured for the same output mode. The Fast Output bit should not be altered during data processing. If it is altered the data output of the cascade will be undefined until new input data has flushed through all stages of the cascade. If the coefficient size is 4 bits there is no difference between the fast and normal modes.

SCR[10]	Output mode
0	Normal*
1	Fast

- **Coefficient Size**

SCR[9–8]	Default: 1 1

Defines size of coefficient used, in terms of word width. This also determines the minimum interval between data input cycles and thus the data throughput of the IMSA100. The Coefficient Size bits should not be altered during data processing. If they are altered the data output of the cascade will be undefined until new input data has flushed through all stages of the cascade.

In each mode the coefficient data is the least signifcant bits of the 16 bit word; e.g. in 4 bit mode, a two's complement number should be programmed into bits 0–3 of the 16 bit register. The remaining bits 4–15 are ignored.

SCR[9–8]	Coefficient size	Data input interval
0 0	4 bits	2 cycles
0 1	8 bits	4 cycles
1 0	12 bits	6 cycles
1 1	16 bits	8 cycles *

- **Reserved**

SCR[7–6]	Default : 0 0

These locations are reserved. The user should write 0,0 to these locations to maintain compatability with future products. The value read from this location is undefined.

- **Reserved**

SCR[3]	Default: 0

This location is reserved. The user should write 0 to this location to maintain compatability with future products. The value read from this location is undefined.

- **Output Word Selection**

SCR[5–4]	Default: 1 0

These bits determine the 24 bit wide field selected from the 36 bit wide output of the multiplier accumulator array (bit positions numbered 0 to 35).

The word selected will be rounded and sign extended before being output. Note that ranges '10' and '11' imply sign extension of the result.

The Output Word Selection bits should not be altered during data processing. If they are altered the data output of the cascade will be undefined until new input data has flushed through all stages of the cascade.

SCR[5–4]	Field
0 0	[7–30]
0 1	[11–34]
1 0	[15–38] *
1 1	[20–43]

- **Continuous Swap**

SCR[2]	Default: 0

The Continuous Swap bit selects whether the two banks of coefficient registers are automatically exchanged after each data input and computation cycle or if individual bank swaps occur under the direction of the Bank Swap bit in the Active Control Register, ACR[0]. SCR[2] should not be set if a bankswap has been requested (by setting ACR[0]) and is still pending.

SCR[2]	Swap Mode
0	Swap on asserting ACR[0] *
1	Swap after end of each input cycle

- **Input Data Source**

SCR[1]	Default: 0

The data source for the multiplier accumulator array can come from one of two sources, selected by SCR[1]. Data can either be input from the **DIN** port or it can be written into the Data Input Register via the memory interface. See also the following section.

SCR[1]	Data Source
0	From **DIN** port *
1	From DIR

- **Master not Slave**

SCR[0]	Default: 0

The Master not Slave bit selects whether the IMSA100 samples the **GO** input to determine the start of a data input cycle (slave mode), or drives the **GO** pin when data is written to the DIR (master mode). If input data is supplied through the DIR one IMSA100 in the cascade should be configured as a master. If data is supplied to the **DIN** port by an external data source all the IMSA100s in the cascade should be configured as slaves and **GO** should be driven by an external system. Note that an illegal mode results if SCR[1] is 0 and SCR[0] is 1; i.e. a master cannot obtain data from the **DIN** port.

SCR[0]	Mode
0	Slave *
1	Master

4.4 Active control register

- **Cascade Adder Overflow**

ACR[2]	Default: 0

If previously armed this status bit will be set if the addition of the 24 bit words output by the 24 from 36 bit selector (on the output of the multiply accumulator array) and the cascade shift register causes an arithmetic overflow.

The **$\overline{\text{ERROR}}$** pin will be driven low while this or any other error condition is active. This error bit and the error condition can be cleared by writing a zero to ACR[2], provided the data in the adder is no longer in error. After clearing this error bit the error bit should be armed (by writing a one to ACR[2]) to ensure that any future error is detected. See **$\overline{\text{ERROR}}$** section.

- **Selector Overflow**

ACR[1]	Default : 0

If previously armed this status bit will be set if the 24 bit output range of the selector does not include all the significant binary digits in the 36 bit result generated by the multiply accumulator array. The **$\overline{\text{ERROR}}$** pin will be driven low while this or any other error condition is active. This error bit and the error condition can be cleared by writing a zero to ACR[1]. After clearing this error bit the error bit should be armed (by writing a one to ACR[1]) to ensure that any future error is detected. See **$\overline{\text{ERROR}}$** section.

- **Initiate Bank Swap**

ACR[0]	Default : 0

Writing a one into this control bit requests an exchange of data between the Current and Update Coefficient Registers. The bank swap will occur as soon as the current computation cycle is completed, or on the next clock cycle if the IMSA100 is idle. This control bit is cleared to zero by the IMSA100 when the bank swap is complete. No access should be made to either set of coefficient registers while a bank swap is in progress. ACR[0]

should not be set if SCR[2] is already set. For a detailed description of the behaviour see the bank-swap and coefficient access timing diagrams.

4.5 Test control register

- **Examine Full Output Word**

TCR[2]	Default: 0

This bit overrides the output word selection normally made by bits SCR[5–4]. The output word selection determines the 24 bit wide field selected from the 36 bit wide output of the multiply accumulator array (bit positions numbered 0 to 35). When TCR[2] is set to '1' the output word selection is bits '-1' to 22, where bit '-1' is set to zero. The output word selection should not be altered during data processing. If altered the data output of the cascade will be undefined until new input data has flushed through all stages of the cascade.

TCR[2]	Field
0	Set by SCR[5–4] *
1	[–1 to 22]

- **Reserved**

TCR[1]	Default: 0

This location is reserved for test purposes. For normal operation the user should write 0 to this location.

- **Reserved**

TCR[0]	Default: 0

This location is reserved for test purposes. For normal operation the user should write 0 to this location.

5. DEVICE APPLICATIONS

The IMS A100 can be used in a variety of different applications requiring high performance computation. Some of these are described below, and are covered in detail in the IMS A100 Application Note series.

5.1 Filtering and adaptive filtering

The IMS A100 device can be used to implement high speed FIR and IIR digital filters. The maximum sampling frequency of the input signal ranges between 2.125MHz and 15MHz, depending on the coefficient word length and speed variant that has been selected.

The continuous bank swap mode allows a single device to filter complex (I & Q) data streams. High speed random access coefficient registers enable high performance adaptive filters and equalisers to be realised with minimal complexity.

The cascadability of the device enables FIRs of greater than 32 stages to be constructed, with no degradation in data throughput.

5.2 Convolution and correlation

The IMS A100 is the first single-chip digital correlator capable of highly accurate computation of correlation and convolution functions (16-bit coefficients, 16-bit data and 36-bit accumulation). These functions have applications in matched filtering, noise reduction and pulse compression in communication, radar and sonar systems. For correlations and convolutions involving a large number of data points, devices can be cascaded to several thousand stages with careful design. Alternatively, it is possible to use algorithms which allow decomposition of long correlation and convolutions into several smaller ones, which can then be carried out by a single or smaller number of devices.

5.3 Matrix multiplication

The architecture of the IMS A100 allows very high speed fixed point matrix multiplication. In this application the columns of the multiplier matrix are circulated as inputs to the chip while the coefficients are programmed in a suitable manner with the elements of the multiplicant matrix. Larger matrices can be handled by either cascading several chips or by decomposing the matrices into smaller ones.

5.4 Fourier transforms

Two algorithms, namely the Prime Number Transform (PNT) and the Chirp-Z Transform (CZT), can be used to perform high speed Fourier transforms using IMS A100s. The Fourier transform of long data sequences can be evaluated either by using cascaded IMS A100s or by using decomposition algorithms to convert a long transform into a number of short transforms (e.g. <32 points). These short transforms can then be carried out using the IMS A100s and a host processor.

The speed of transform can be traded off against the number of chips employed. Any microprocessor with a standard memory interface could be used to handle intermediate results and to control the overall system. Two IMS A100s can be used to perform a transform of about 1000 points in around

1ms to 2ms using look-up ROMs for address generation and high speed DSP controllers, or 5ms to 10ms using a microprocessor as the controller. More IMS A100s can be used if higher performance is required.

5.5 Waveform synthesis

The programmability of this digital transversal filter allows the IMS A100 to be used for flexible waveform generation and synthesis, by exploiting the ability to change coefficients randomly, quickly and simply. Such a configuration could be attractive for PC based synthesisers, as the chip can generate very accurate high bandwidth signals.

5.6 General purpose accelerator

By attaching one or more IMS A100s to any computer with DMA capability, a useful accelerator can be constructed, capable of handling all of the above applications without reconfiguration. The cascadability of the device enables users to add IMS A100s as required for extra processing performance, with minimal impact on the driving software.

6. ELECTRICAL SPECIFICATION

The IMS A100 is available in several temperature variants and the electrical characteristics of each are described in this section. When no variant is identified the information refers to all variants.

6.1 DC electrical characteristics

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Min.	Typ.	Max.	Units	Note (1)
V_{CC}	DC supply voltage	0		7.0	V	2,3
V_I, V_O	Voltage on input and output pins	-1.0		V_{CC}+0.5	V	2,3
T_{stg}	Storage temperature	-65		150	°C	2
T_A	Temperature under bias	-55		125	°C	2
P_{Dmax}	Power dissipation			2.0	W	2

Notes

1 All voltages are with respect to **GND**.

2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedence circuit. Unused inputs should be tied to an appropriate logic level such as V_{CC} or **GND**.

DC OPERATING CONDITIONS

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes (1)
V_{CC}	DC supply voltage	4.5 4.75		5.5 5.25	V V	4 5
V_{IH}	Input Logic '1' Voltage CLK	4.0		V_{CC}+0.5	V	2
	Input Logic '1' Voltage $\overline{RESET}$	2.4		V_{CC}+0.5	V	2
	Input Logic '1' Voltage other pins	2.0		V_{CC}+0.5	V	2
V_{IL}	Input Logic '0' Voltage { CLK}	-0.5		0.5	V	2
	Input Logic '0' Voltage $\overline{RESET}$	-0.5		0.8	V	2
	Input Logic '0' Voltage other pins	-0.5		0.8	V	2
T_A	Ambient Operating Temperature	0 -40		70 85	°C °C	3,4 3,5

Notes
1 All voltages are with respect to **GND**. All **GND** pins must be connected to **GND**.
2 Input signal transients up to 10 ns wide, are permitted in the voltage ranges (**GND** - 0.5 V) to (**GND** - 1.0 V) and V_{CC} + 0.5 V to V_{CC} + 1.0 V.
3 400 linear ft/min transverse air flow.
4 IMS A100-G21S.
5 IMS A100-G21I

DC CHARACTERISTICS

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes (1,2)
V_{OH}	Output Logic '1' Voltage	2.4		V_{CC}	V	4
V_{OL}	Output Logic '0' Voltage	0		0.4	V	5
I_I	Input current @ GND<VI<VCC			±10	µA	
I_{OZ}	Tristate output current @ GND<V_I<V_{CC}			±10	µA	
I_{CC}	Average power supply current			360	mA	3

Notes
1 All voltages are with respect to **GND**.All **GND** pins must be connected to **GND**.
2 Parameters measured over variants full voltage and temperature operating range.
3 Power dissipation is application dependent and varies with output loading. The maximum given here is for worst case data patterns and activity on all interfaces, with no DC load on outputs.
4 **OUTRDY, D_{OUT}**: I_{OUT} ≤ -4.4 mA; **$\overline{ERROR}$** is open collector; other outputs: I_{OUT} ≤ -5.5 mA.
5 **OUTRDY, D_{OUT}**: I_{OUT} ≤ 4.4 mA; ERROR: I_{OUT} ≤ 5.5 mA; other outputs: I_{OUT} ≤ 5.5 mA.

CAPACITANCE

Pin	Min.	Typ.	Min.	Units	Notes
CLK		12		pF	1,2
All other pins		5		pF	1,2

Notes
1 This parameter is supplied for engineering guidance and is not guaranteed.
2 T_A = 25°C , f=1MHz.

6.2 Thermal Characteristics

PIN GRID ARRAY THERMAL CHARACTERISTICS

Symbol	Parameter	Min	Typ.	Max	Units	Notes
θ JA	Junction to ambient thermal resistance			35	°C/W	1,2

Notes
1 Measured at 400 linear ft/min transverse air flow.
2 This parameter is sampled and not 100% tested.

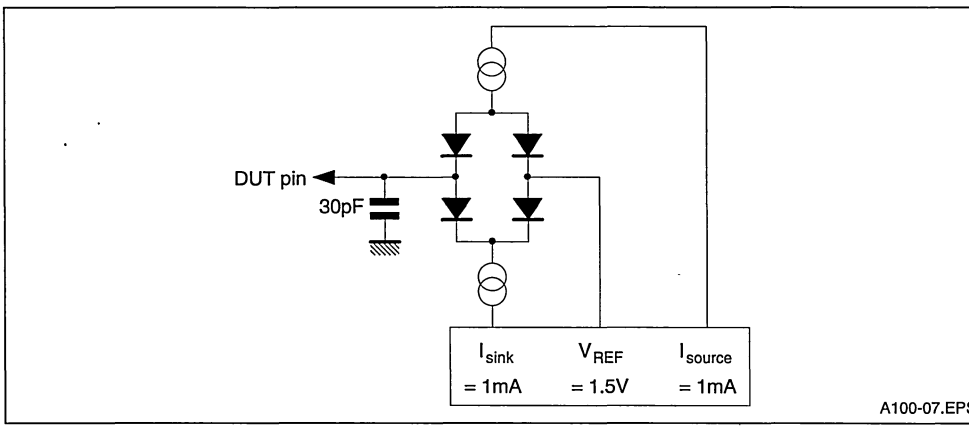
6.3 AC Timing Characteristics

AC TEST CONDITIONS

Output loads (except output turn-off tests)

Pin	Device mode	Load	Unit
GO	Master	20	pF
DOUT, OUTRDY	Fast output	15	pF
DOUT, OUTRDY	Normal output	30	pF
All other outputs	All modes	30	pF

Figure 6 : Output Load (Output Turn-Off tests)



TIMING REFERENCE LEVELS

Pin	Reference levels	Notes
INPUTS	0.8V, 2.0V	1
CLK	0.5V, 4.0V	
OUTPUTS	0.4V, 2.4V	2,3
OUTPUTS	±100mV change from previous steady output voltage	4

Notes

1 Except **CLK**.
2 Output continuously driven.
3 Timings are tested using VOL=0.8V and with a suitable allowance for the time taken for the output to fall from 0.8V to 0.4V
4 Output turn-off tests.

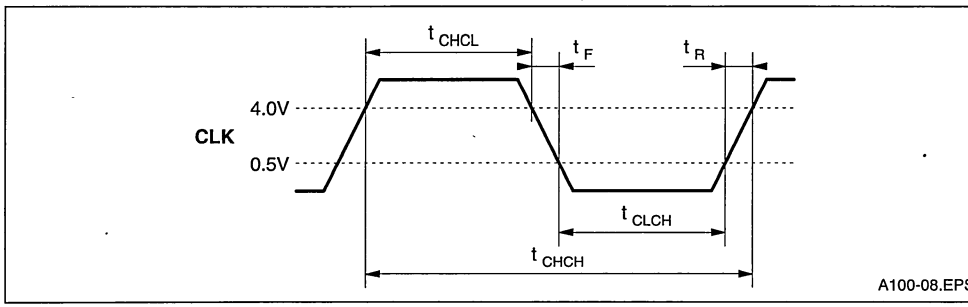
CLOCK

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{CHCL}	Clock pulse width high	19			ns	
t_{CLCH}	Clock pulse width low	19			ns	
t_{CHCH}	Clock period	48			ns	
t_R	Clock rise time	0		50	ns	1
t_F	Clock fall time	0		50	ns	1

Note :

1 Clock input transitions should be monotonic between the input thresholds of 0.5 V and 4.0 V.

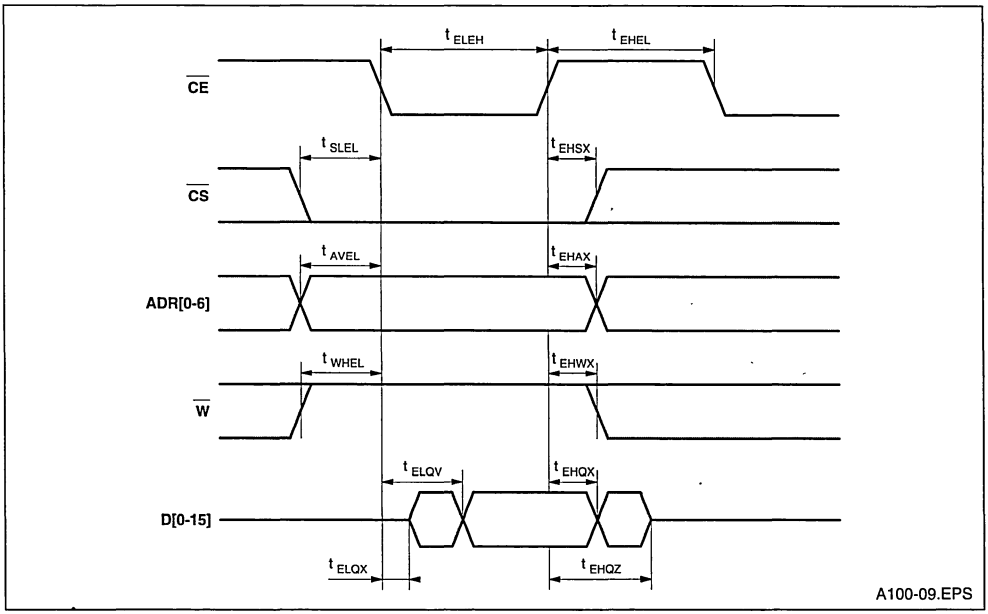
Figure 7



MEMORY INTERFACE READ CYCLE

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{ELEH}	$\overline{CE}$ pulse width low	60			ns	
t_{EHEL}	$\overline{CE}$ pulse width high	50			ns	
t_{SLEL}	$\overline{CS}$ setup time	15			ns	
t_{EHSX}	$\overline{CS}$ hold time	5			ns	
t_{AVEL}	Address setup time	15			ns	
t_{EHAX}	Address hold time	5			ns	
t_{WHEL}	Read Command setup	15			ns	
t_{EHWX}	Read Command hold	5			ns	
t_{ELQX}	Output turn on delay	0			ns	
t_{ELQV}	Read data access			60	ns	
t_{EHQX}	Read data hold	0			ns	
t_{EHQZ}	Output turn off delay			25	ns	

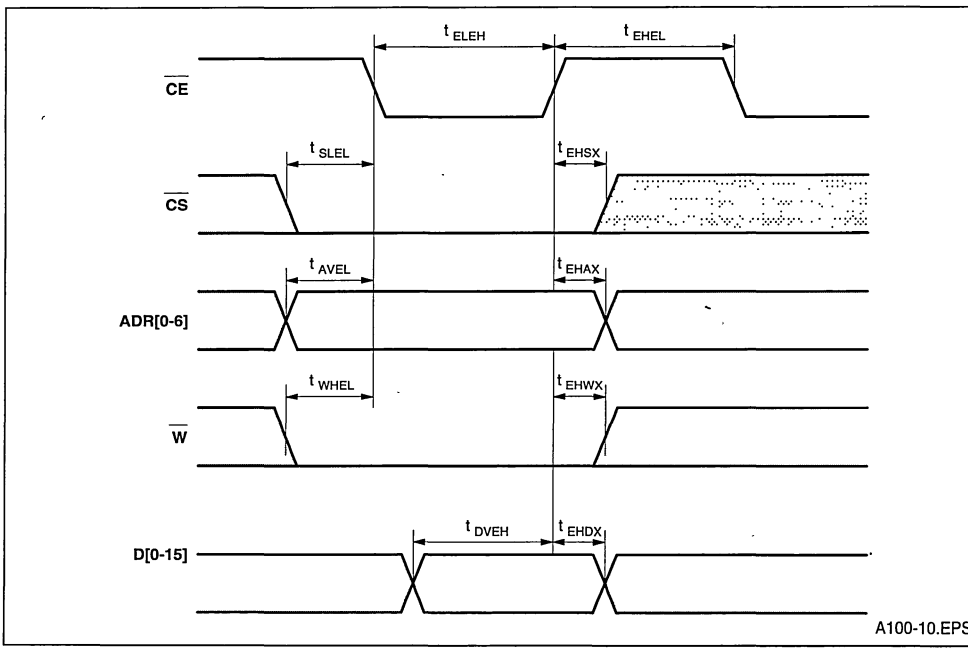
Figure 8



MEMORY INTERFACE WRITE CYCLE

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{ELEH}	$\overline{CE}$ pulse width low	50			ns	
t_{EHEL}	$\overline{CE}$ pulse width high	50			ns	
t_{SLEL}	$\overline{CS}$ setup time	15			ns	
t_{EHSX}	$\overline{CS}$ hold time	5			ns	
t_{AVEL}	Address setup time	15			ns	
t_{EHAX}	Address hold time	5			ns	
t_{WLEL}	Write Command setup	15			ns	
t_{EHWX}	Write Command hold	5			ns	
t_{DVEH}	Write data setup	45			ns	
t_{EHDX}	Write data hold	5			ns	

Figure 9



STATIC READ ACCESSES TO DOL AND DOH REGISTERS

Certain applications require to read results from the IMS A100 at high speeds. To ensure full system performance it may be necessary to read results from the DOL and DOH registers using a continuous 'static' access rather than using the normal clocked access.

During static access the $\overline{CE}$ signal is held low continuously. Under this condition it is possible to monitor either DOL or DOH continuously to observe new output words as they become available or alternatively to switch between DOL and DOH without the restriction of having to sequence $\overline{CE}$.

Symbol	Parameter	Min	Typ.	Max	Units	Notes
t_{AVQV}	Address access time			75	ns	1
t_{CHQV}	Data input access time			τ+75	ns	2
t_{ELQV}	$\overline{CE}$ access time			60	ns	3
t_{AXQX}	Data hold after address change	0			ns	
t_{CHQX}	Data hold after new data input	τ+0			ns	2
t_{EHQX}	Data hold after end of read	0			ns	

Notes

1 The address access time is specified for address transitions between decimal 74 (DOL register) and decimal 75 (DOH register) only.

2 The parameter τ describes the time taken from the input of a data word to that data word first affecting the most significant word (MSW) output. This is the time at which the DOL and DOH registers are updated.
The duration of τ depends on the coefficient size selected and whether fast or normal output is selected.

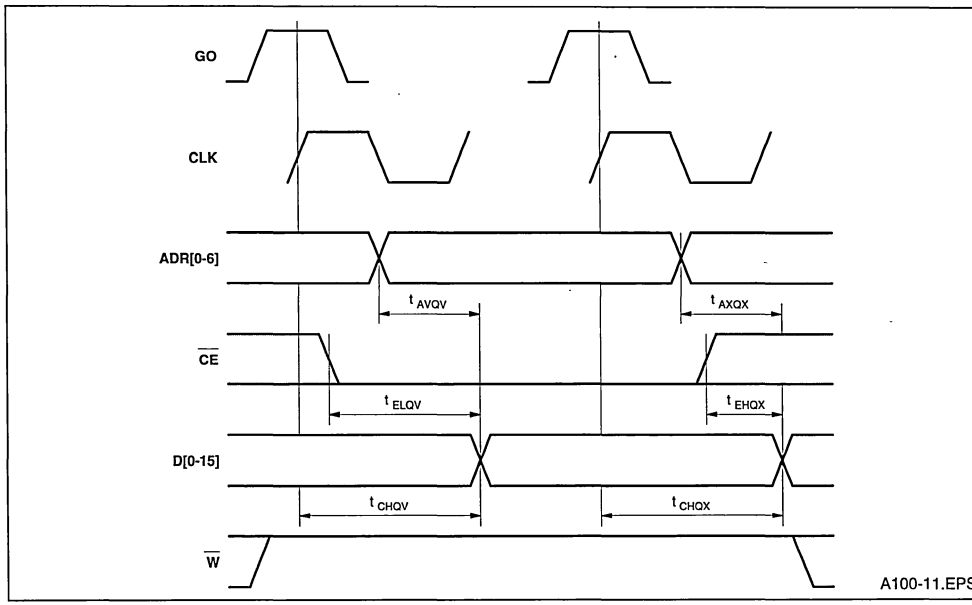
Coefficients	Output mode	τ time
4 bit	Fast	8 CLK cycles
8 bit		10 CLK cycles
12 bit		12 CLK cycles
16 bit		14 CLK cycles

Coefficients	Output mode	τ time
4 bit	Normal	Not defined
8 bit		11 CLK cycles
12 bit		14 CLK cycles
16 bit		17 CLK cycles

N.B. The data value read from either DOL or DOH will change as new results are computed by the device.

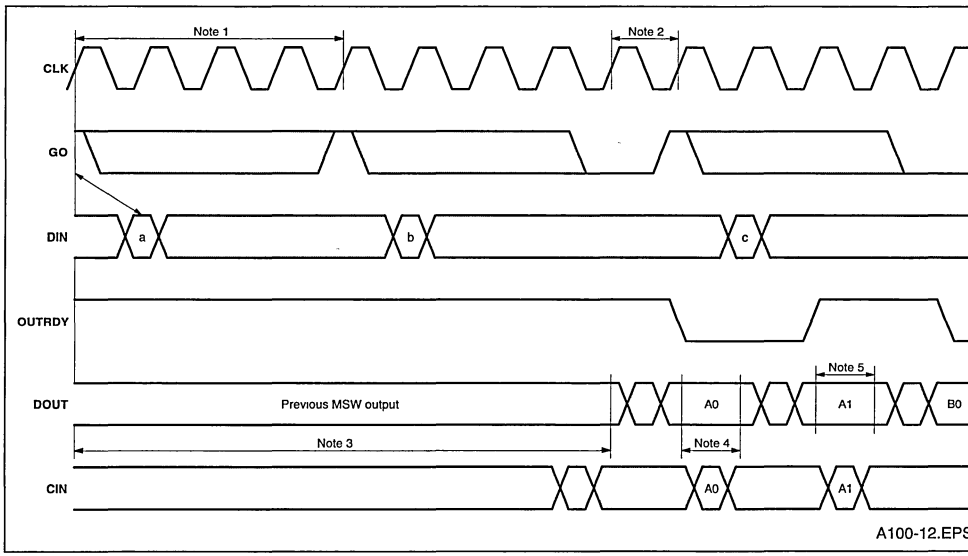
3 This parameter is the normal read access time for reading any register through the microprocessor interface. In the special case of performing reads from only DOL and DOH any number of reads from these registers can be made with $\overline{CE}$ held low continuously.
It is required that a static access (as described above) should commence like a normal clocked, random, read access to either DOL or DOH. That is **ADDRESS**, $\overline{CS}$ and W should be established with setup times to $\overline{CE}$ specified for a normal read access.
During a DOL/DOH static access sequence accesses to locations other than DOL and DOH are undefined.

Figure 10



TYPICAL SEQUENCE - 8 BIT COEFFICIENTS, NORMAL OUTPUT

Figure 11



Notes

1 The minimum period between sampling the **GO** input is four clock cycles for 8 bit coefficients, see the table below for the other cases.

2 After the minimum period described in note 1 has elapsed **GO** is sampled on every rising edge of **CLK** until **GO** is high.

3 The delay from an output being initiated by **GO** to the output completing its previous output sequence and starting the new output sequence is 8 clock cycles for 8 bit coefficients, see the table below for the other cases.

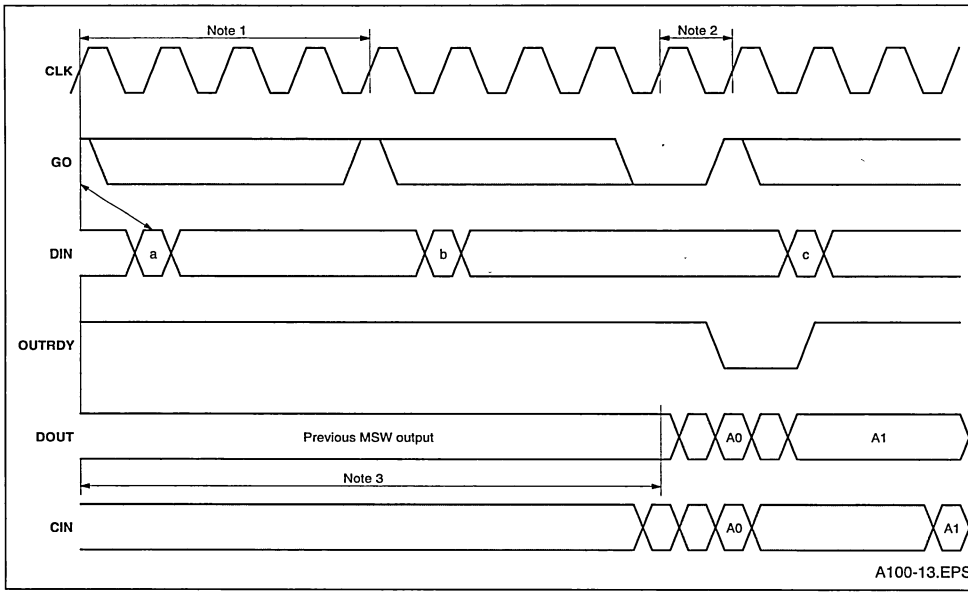
4 The least significant word is available at the output across one complete **CLK** cycle for the 8 bit coefficient, normal output case, see the table below for the other cases.

5 The most significant word is available for the minimum period described in note 4, but will be extended by a clock cycle for each additional idle cycle inserted between data inputs.

Coefficients	**Min. Output Period** note 1	**Delay To Output** note 3	**Min. LSW Output Duration** notes 4 and 5
4 bit	2 CLK cycles	6 CLK cycles	Undefined, no normal output
8 bit	4 CLK cycles	8 CLK cycles	1 CLK cycle
12 bit	6 CLK cycles	10 CLK cycles	2 CLK cycles
16 bit	8 CLK cycles	12 CLK cycles	3 CLK cycles

TYPICAL SEQUENCE - 8 BIT COEFFICIENTS, FAST OUTPUT

Figure 12



Notes

1 The minimum period between sampling the **GO** input is four clock cycles for 8 bit coefficients, see the table below for the other cases.

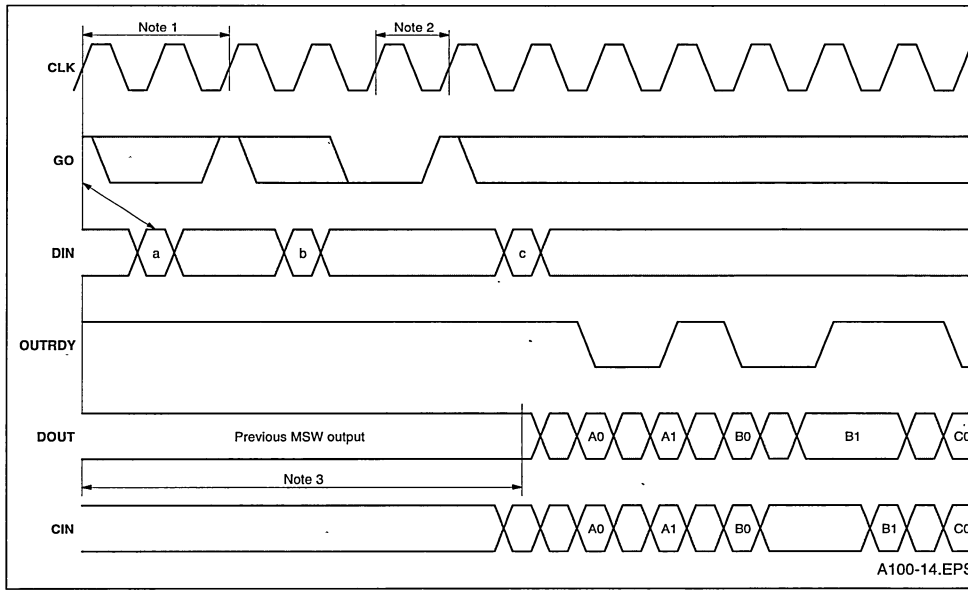
2 After the minimum period described in note 1 has elapsed **GO** is sampled on every rising edge of **CLK** until **GO** is high.

3 The delay from an output being initiated by **GO** to the output completing its previous output sequence and starting the new output sequence is 8 clock cycles for 8 bit coefficients, see the table below for the other cases.

Coefficients	**Min. Output Period** note 1	**Delay To Output** note 3
4 bit	2 CLK cycles	6 CLK cycles
8 bit	4 CLK cycles	8 CLK cycles
12 bit	6 CLK cycles	10 CLK cycles
16 bit	8 CLK cycles	12 CLK cycles

TYPICAL SEQUENCE - 4 BIT COEFFICIENTS

Figure 13



Notes

1 The minimum period between sampling the **GO** input is two clock cycles for 4 bit coefficients, see the table below for the other cases.

2 After the minimum period described in note 1 has elapsed **GO** is sampled on every rising edge of **CLK** until **GO** is high.

3 The delay from an input being initiated by **GO** to the output completing its previous output sequence and starting the new output sequence is 6 clock cycles for 4 bit coefficients, see the table below for the other cases.

Coefficients	**Min. Output Period** note 1	**Delay To Output** note 3
4 bit	2 CLK cycles	6 CLK cycles
8 bit	4 CLK cycles	8 CLK cycles
12 bit	6 CLK cycles	10 CLK cycles
16 bit	8 CLK cycles	12 CLK cycles

NORMAL OUTPUT TIMING–8 BIT COEFFICIENT CASE SHOWN

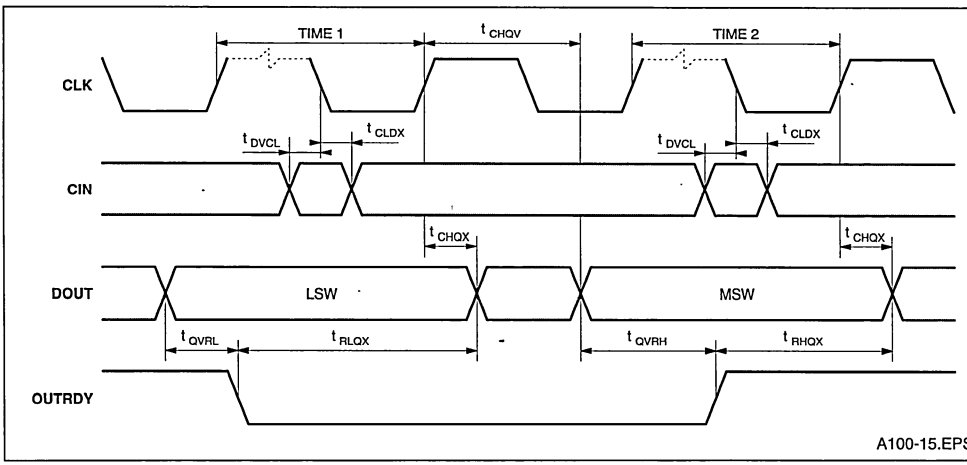
Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{CHQV}	CLK high to DOUT valid delay			36	ns	
t_{CHQX}	DOUT hold time after CLK high	2			ns	
t_{QVRL}	DOUT to OUTRDY low lead	15			ns	
t_{RLQX}	DOUT hold time after OUTRDY low	10			ns	1
t_{QVRH}	DOUT to OUTRDY high lead	15			ns	
t_{RHQX}	DOUT hold time after OUTRDY high	10			ns	1,2
$t_{IME}1$	LSW output duration	1		3	t_{CHCH}	1
$t_{IME}2$	MSW output duration	1		3 ,t_{CHCH}	1,2	
t_{DVCL}	CASIN setup time to CLK low	10			ns	
t_{CLDX}	CASIN hold time from CLK low	10			ns	

Notes

1 This parameter is determined by the coefficient size in use. The minimum value given is correct for 8 bit coefficients. This parameter is extended by 1 (or 2) periods of **CLK** if 12 (or 16) bit coefficients are used. This mode of operation is not defined if 4 bit coefficients are used.

2 These parameters are extended by one tCHCH for each idle cycle inserted between data input sequences

Figure 14



FAST OUTPUT TIMING–4 BIT COEFFICIENT CASE SHOWN

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{CHQV}	CLK high to DOUT valid delay			36 22	ns ns	1 1
t_{CHQX}	DOUT hold time after CLK	2			ns	2
t_{QVRL}	DOUT to OUTRDY low lead				ns	
t_{RLQX}	DOUT hold time after OUTRDY low	10			ns	
t_{QVRH}	DOUT to OUTRDY high lead	5			ns	1
t_{RHQX}	DOUT hold time after OUTRDY high	10			ns	2
t_{DVCH}	CASIN setup time to CLK high	10			ns	
t_{CHDX}	CASIN hold time to CLK high	0			ns	3

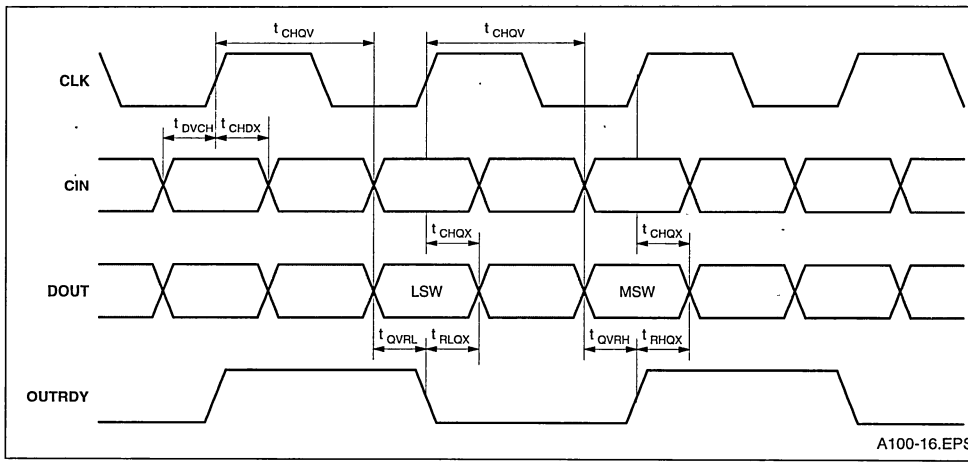
Notes

1 These parameters assume that each **DOUT** signal is loaded with a maximum of 15 pF.

2 t_{CHQX} and t_{RHQX} for the MSW are shown here for the case where 4 bit coefficients are being used. In the other cases (8, 12 and 16 bit coefficients) the MSW is available for an additional 2, 4 or 6 **CLK** periods. In all cases the MSW will be available for an additional period of **CLK** for each idle cycle inserted between data input sequences.

3 Not tested. Guaranteed by design.

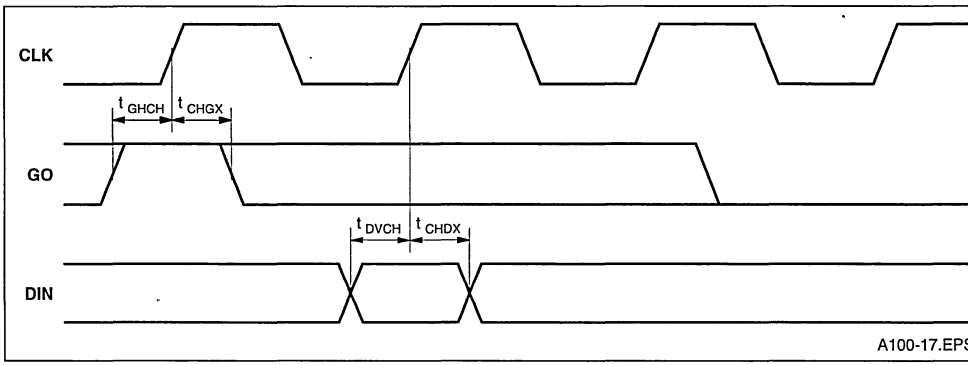
Figure 15



EXTERNAL GO AND DATA INPUT TIMING

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{GHCH}	GO setup time	10			ns	
t_{CHGX}	GO hold time	5			ns	
t_{DVCH}	DIN setup time	30			ns	
t_{CHDX}	DIN hold time	5			ns	

Figure 16



MASTER GENERATED GO

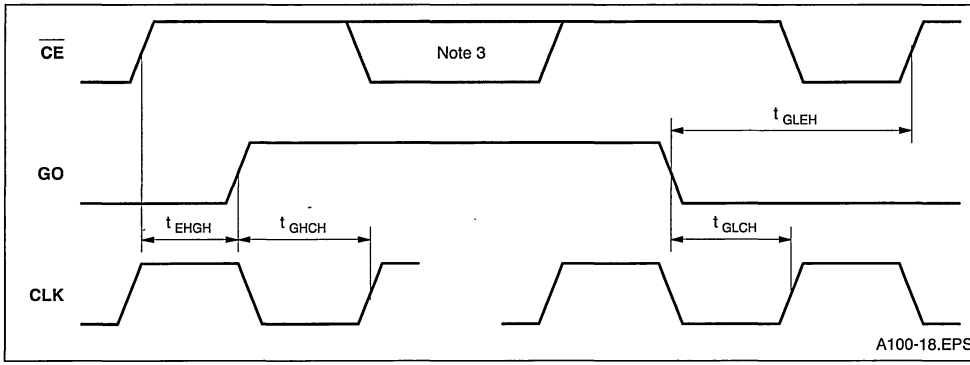
Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{EHGH}	Write to DIR to GO high delay	25			ns	1
t_{GHCH}	GO high before GO sampled	10			ns	2
t_{GLEL}	GO low to write to DIR	0			ns	
t_{GLCH}	GO low before GO next sampled	10			ns	2

Notes

1 The maximum delay from a write to the DIR to **GO** going high is 2 * t_{CHCH} + 50 ns.

2 This parameter assumes the capacitive load on **GO** is less than 20 pF. **GO** is specified so that one master IMS A100 can drive three slave IMS A100s without buffering.

Figure 17



BANKSWAP TIMING

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{EHBH}	ACR[0] set to BUSY high delay			55	ns	
t_{CHBL}	BUSY hold after bankswap			50	ns	
t_{CHEH}	ACR[0]=0 hold after last input	20			ns	3
t_{EHCH}	ACR[0]=1 setup to next input	10			ns	3

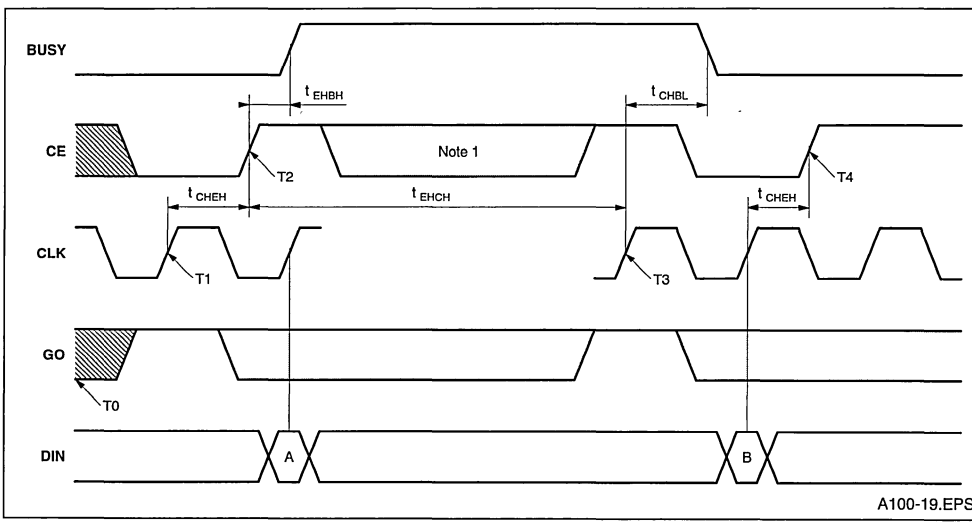
Notes

1 The activity on $\overline{CE}$ shown is for writing ACR[0]=1. During the period **Note 1** it may be possible to access other registers (subject to their own access constraints).

2 For small t_{EHCH}, **BUSY** may only occur for a short time or not occur at all.

3 If t_{CHEH} or t_{EHCH} is exceeded then bankswap may be synchronised to the previous or next input cycle.

Figure 18



The bankswap timing diagram shows how successive data samples (A and B) can be processed by different sets of coefficients by causing a bankswap to occur between the input of sample A and sample B.

The sequence of events is as follows:

T0 No bankswap pending.

T1 GO sampled and found to be high, thus initiating input of data sample A.

T2 Bankswap requested by writing ACR[0]=1. If the minimum timing requirement, t_{CHEH}, from T1 to T2 is not met it is possible (but not guaranteed) that the bankswap requested at T2 will occur immediately and thus affect the processing of data sample A.

T3 Bankswap occurs on the first rising edge of **CLK** upon which **GO** is sampled (without reference to the state of **GO**). If the minimum timing requirement, t_{EHCH}, from T2 to T3 is not met it is possible (but not guaranteed) that the bankswap requested at T2 will not occur at T3 but at the next sampling of **GO**.

T4 This is the earliest time at which another bankswap can be requested.

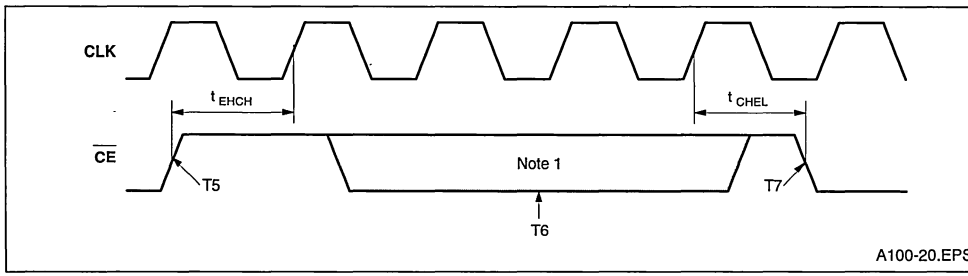
COEFFICIENT ACCESS TIMING

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_{EHCH}	End coefficient access before bankswap	0			ns	
t_{CHEL}	Start coefficient access after bankswap	0			ns	

Notes

1 During this period accesses may be made to registers other than the coefficient registers (subject to their own access constraints).

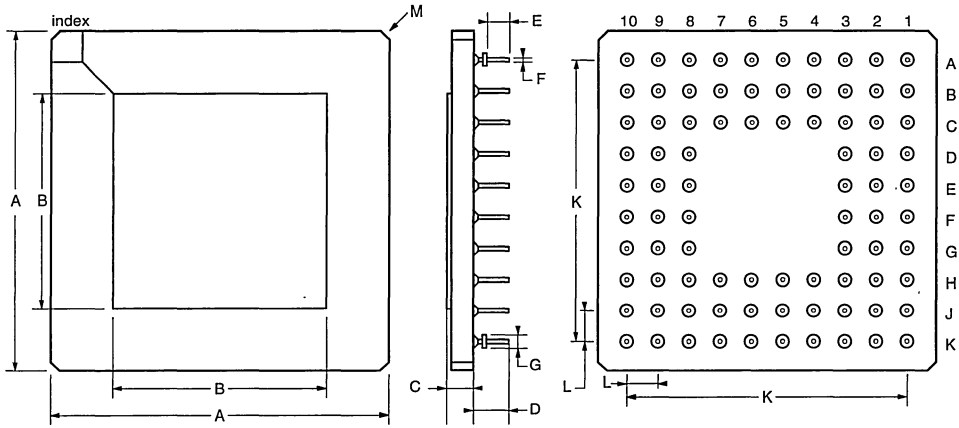
Figure 19



If a bankswap (caused by setting either ACR[0]=1 or SCR[2]=1) occurs at the **GO** sampling point T6, then no access should be made to the coefficient registers between T5 and T7.

PACKAGE MECHANICAL DATA

84 PINS - GRID ARRAY PACKAGE



DIM	Millimètres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	26.924	±0.254	1.060	±0.010	
B	17.019	±0.127	0.670	±0.005	
C	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.025	0.018	±0.001	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
M	0.508		0.020		Chamfer
Package weigth is approwimately 2.2 grams					

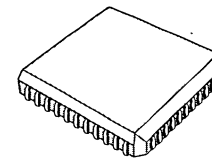
PGA84.TBL

DATASHEETS

IMAGE CODING DEVICES

2-D DISCRETE COSINE TRANSFORM IMAGE PROCESSOR

- 8 X 8 TRANSFORM SIZE.
- 8 X 8 DCT CALCULATION TIME = 3.2µs.
- DC TO 20 MHZ PIXEL RATE.
- 9-BIT ADD/SUBTRACT INPUT.
- 12-BIT INPUT/OUTPUT.
- 14-BIT FIXED COEFFICIENTS.
- MULTIFUNCTION CAPABILITY (DCT, IDCT, FILTER).
- FULL INTERNAL PRECISION, FOR EACH DIMENSION.
- FULLY SYNCHRONOUS INTERFACE.
- HIGH SPEED CMOS IMPLEMENTATION.
- TTL COMPATIBLE.
- SINGLE +5V ± 10%.
- POWER DISSIPATION < 1.5 WATT.
- 44 PIN PLASTIC PACKAGE.



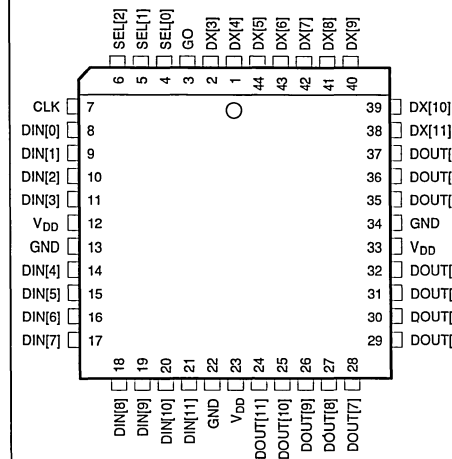
PLCC44
(Plastic Chip Carrier)

ORDERING INFORMATION

Designation	Package	Clock speed
IMS A121-J20S	PLCC44	20MHz

A121-01.TBL

PIN CONNECTIONS



Note
All **Vcc** pins **must** be connected to the 5 Volt power supply.
All **GND** pins **must** be connected to ground.

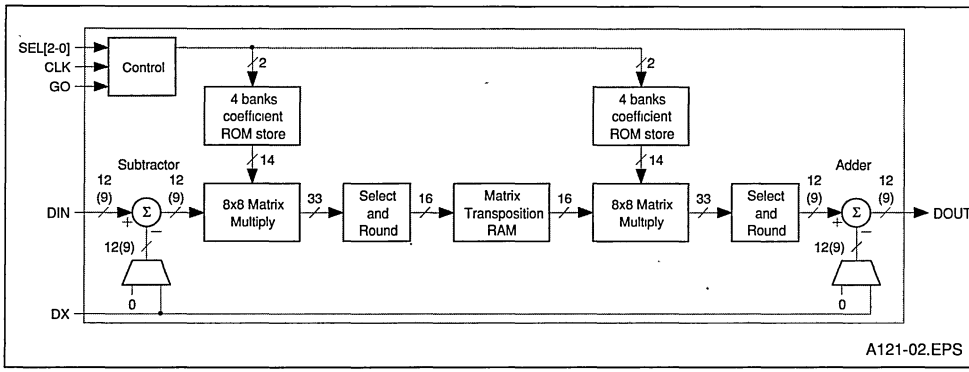
A121-01.EPS

DESCRIPTION

The IMS A121 is a device for computing the Discrete Cosine Transform (DCT & IDCT). It will also function as a 2-D linear filter or perform matrix transposition. These 4 functions operate on blocks of data with a fixed size of 64 samples (8 × 8). The IMS A121 has other functions aimed specifically at the implementation of video codecs; on-chip subtraction and addition functions may be selected to reduce system chip count.

The main computation is performed by two identical multiplication arrays, each of which perform an 8 x 8 matrix multiplication in 64 cycles, with no internal rounding. The DCT/filter coefficients (14-bit) are stored in 4 banks of fixed ROM. The intermediate 8 × 8 matrix result is rounded to 16 bits and stored in the transposition RAM between each multiplication array. The device is fully pipelined with data sampled on the input at the clock frequency and the resultant output appearing 128 clock cycles later.

Figure 1



1. OVERALL DEVICE OPERATION

The IMS A121 is a device for computing the Discrete Cosine Transform (DCT) and the Inverse Discrete Cosine Transform (IDCT). It can also perform a simple low-pass filter operation.

The IMS A121 processes blocks of data which are 64 samples long and represent an 8 × 8 matrix. Data is sampled on the **Din** port every cycle and data is output every cycle on the **Dout** port.

The **GO** signal is used to indicate the start of a block. When it is sampled high the data on the **Din** port is the first sample of the block. The mode select signals **SEL[2-0]** are sampled at the same time. The remainder of the block of data is sampled on the **Din** port for the subsequent 63 cycles and during this time the **GO** signal and the **SEL** port are ignored. Each consecutive group of eight samples is treated as a column, eight such columns making a block.

The computation is in two stages, between which the block of 64 intermediate samples is stored in the transposition RAM. The transposition RAM serves a dual function of storing the intermediate results and transposing the data from column order into row order. This permits the two matrix computation elements to be identical although the first stage does the column computations and the second stage does the row computations.

Data is output on the **Dout** port in blocks of 64 samples. However, each consecutive group of eight samples now represents a row because of the internal transposition of data. The first sample of the block is output on the **Dout** port 128 cycles after the first sample of the block was sampled on the **Din** port.

An auxillary port, **Dx** is provided. The data on the **Dx** port is optionally subtracted from the data on the **Din** port (DCT mode) or added to the output (IDCT mode).

The IMS A121 views input data in column order and (because of the internal transposition) output data in row order. However, this convention is only used to define the arithmetic which the IMS A121 performs. The system in which the IMS A121 is a component may well view the data going into the IMS A121 in row order and the data coming out in column order.

1.1 The fixed ROM coefficients

There are four sets of fixed ROM coefficients, each corresponding to one of the four possible functions the device can perform. The two main functions which the device can perform are the DCT and the IDCT. The other two functions provide assistance for the implementation of a video codec. The filter function is provided at very little overhead because the device is essentially a 2-D filter. The transposition function which is a unity multiplication, enables a simple method of switching out the filter without any external logic.

1.2 Number formats

All numbers input to the IMS A121 are signed integers. The **Din** and **Dout** ports use 12 bit signed integers, while the **Dx** port uses 9 bit signed integers. In both cases the number format is twos complement binary. Little Endian format is assumed throughout, so that, for example, **Din[0]** is the least significant bit of the **Din** port and **Din[11]** the most significant (sign) bit. When a nine bit number is transfered over one of the 12 bit ports the most significant nine bits are used. The lowest three bits of the **Din** port are ignored and the lowest three bits of the **Dout** port will be zero.

1.3 Internal Bit-field Selectors and Rounding

The transforms are implemented by a matrix multiplication with no truncation or rounding. This yields a 33 bit result, with bit-field selectors provided to select the parts of the result which are of interest. 16 bits are selected from the output of the first matrix multiplication, which are stored in the matrix transposition RAM. Either 9 bits or 12 bits are selected from the output of the second matrix multiplication (depending on the selected mode).

Bits below the selected range are discarded although the result is rounded not truncated. This is a simple round towards $+\infty$; if the most significant bit of those bits which have been discarded is set then one is added to the bits which are retained.

1.4 Overflow, Saturation and Clipping

Overflow can occur in the subtraction unit, the two bit-field selectors or the addition unit. Overflow occurs whenever there are insufficient bits in the result to represent the number. When overflow occurs the result is replaced by the most positive or the most negative number which can be represented (depending on the sign of the correct result). The device will normally be used in a feedback system. If either positive or negative overflow occurs, then inaccuracies have been introduced. However, the system will remain stable.

In some of the IDCT modes the output is clipped so that all results are positive and all negative numbers are replaced by zero. This ensures that the output is a valid (8-bit) pixel, between 0 and 255.

1.5 Subtraction with the DCT function

When the IMS A121 is used to perform the DCT, it is possible to enable the on-chip subtraction unit, so that before the DCT the data on the **Dx** port is subtracted from the data on the **Din** port. The data is presented to the **Dx** port at exactly the same time as to the **Din** port.

In DCT mode the data on the **Din** port is a nine bit number (the lowest 3 of 12 bits are ignored). The result of the subtraction is saturated to nine bits before being passed to the matrix multiplier.

1.6 Addition with the IDCT function

When the IMS A121 is used to perform the IDCT, it is possible to enable the on-chip addition unit, so that after the IDCT of the data has been done, the result may be added to the data on the **Dx** port. The timing requires careful consideration because of the latency of the device (128 cycles). The first sample of a block must be presented on the **Dx** port 124 cycles after the first sample was presented to **Din**. The data presented to the **Dx** port should be transposed and is thus in the same order as it will come out of **Dout** four cycles later.

The result of the addition is saturated to nine bits and then clipped so that all negative numbers are replaced by zero. The nine bit result is presented on **Dout[11-3]**, while **Dout[2-0]** will be zero. **Dout[11]** will be zero because all the numbers are positive.

Two modes are provided which perform the IDCT without addition. One of these modes disables the adder completely so that nine bit signed results appear on **Dout**. The other mode does NOT add on the value on the **Dx** port but still clips the result so that only positive values appear on **Dout**.

1.7 Resetting

The IMS A121 does not have a reset pin. At power-on the internal state will be undefined and as a result the first three blocks processed are not guaranteed correct. **GO** must be held low for at least 191 cycles to ensure that when it does go high it is interpreted as the start of a block.

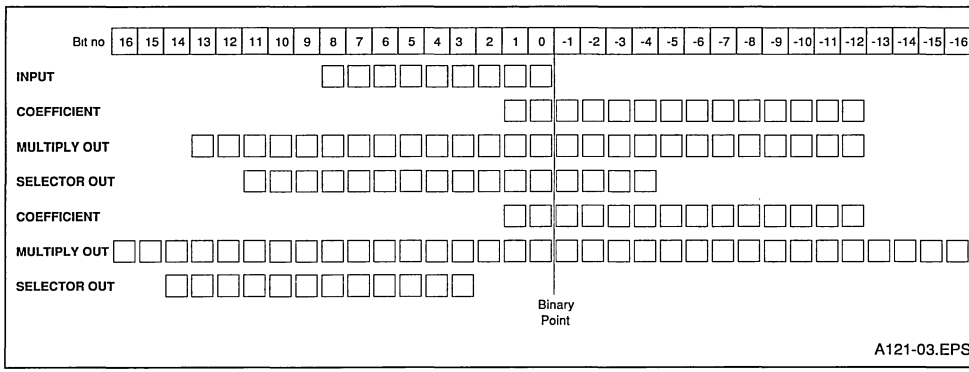
2. DCT FUNCTION

The DCT function is selected when **SEL[2-0]**=000 or 100 (mode 0 or 4).

2.1 Internal number format

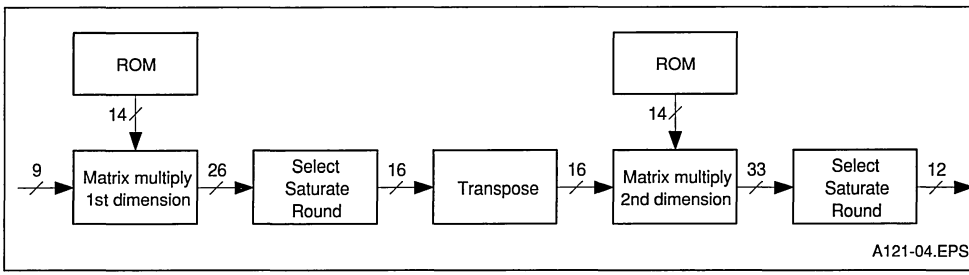
The input for the DCT is a 9 bit signed integer in the range -256 to +255. This is either an external input or the output of the on-chip subtractor depending on **SEL[2-0]**. The input is multiplied in the matrix multiplication array by 14 bit signed fixed point numbers in the range -2 to $(2\text{-}2^{-12})$. The accumulated result of 8 multiply operations is a 26 bit signed integer, the bottom 8 bits of which are rounded (see section 1.3) and the top 2 bits used to saturate the output (see section 1.4). The result of the first matrix multiply is stored as a 16 bit signed integer and the second matrix multiply performed in exactly the same manner, yielding 33 bit results. The output rounds the bottom 19 bits, saturates the top 2 bits giving a 12 bit signed integer in the range -2048 to +2047.

Figure 2 : DCT Internal Number Format



2.2 Internal data flow

Figure 3 : Internal Data Flow



2.3 The mathematical basis for the DCT

The 1 dimensional equation for the DCT is as follows:

Forward transform

$$X(k) = \sqrt{\frac{2}{N}}\, c(k) \sum_{m=0}^{N-1} x(m) \cos\left[\frac{(2m+1)k\pi}{2N}\right] \quad k=0,1,\ldots,N-1$$

where

$$c(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } k=0 \\ 1 & \text{for } k=1\ldots N-1 \end{cases}$$

where x(m) represent the input samples and X(k) is the resulting output. The special case for the IMSA121 is with N=8 and the actual filter coefficients are then calculated. The following equation is used to calculate the actual filter coefficients.

DCT coefficients

$$Coeff_{km} = \sqrt{2}\, c(k) \cos\left[\frac{(2m+1)k\pi}{2N}\right]$$

It should be noticed that the coefficients are $2\sqrt{2}$ times bigger than in the forward transform equation. This means that the output after the 2 dimensional DCT is 8 times too big (The 1 dimensional transform is applied twice giving $(2\sqrt{2})^2$ magnitude increase). This is in accordance with the 3 bit shift of the output data necessary to give the correct 12 bit signed output.

DCT coefficients

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.3870	1.1759	0.7857	0.2759	-0.2759	-0.7857	-1.1759	-1.3870
1.3066	0.5412	-0.5412	-1.3066	-1.3066	-0.5412	0.5412	1.3066
1.1759	-0.2759	-1.3870	-0.7857	0.7857	1.3870	0.2759	-1.1759
1.0000	-1.0000	-1.0000	1.0000	1.0000	-1.0000	-1.0000	1.0000
0.7857	-1.3870	0.2759	1.1759	-1.1759	-0.2759	1.3870	-0.7857
0.5412	-1.3066	1.3066	-0.5412	-0.5412	1.3066	-1.3066	0.5412
0.2759	-0.7857	1.1759	-1.3870	1.3870	-1.1759	0.7857	-0.2759

DCT coefficients (14 bit signed integers)

4096	4096	4096	4096	4096	4096	4096	4096
5681	4816	3218	1130	-1130	-3218	-4816	-5681
5352	2217	-2217	-5352	-5352	-2217	2217	5352
4816	-1130	-5681	-3218	3218	5681	1130	-4816
4096	-4096	-4096	4096	4096	-4096	-4096	4096
3218	-5681	1130	4816	-4816	-1130	5681	-3218
2217	-5352	5352	-2217	-2217	5352	-5352	2217
1130	-3218	4816	-5681	5681	-4816	3218	-1130

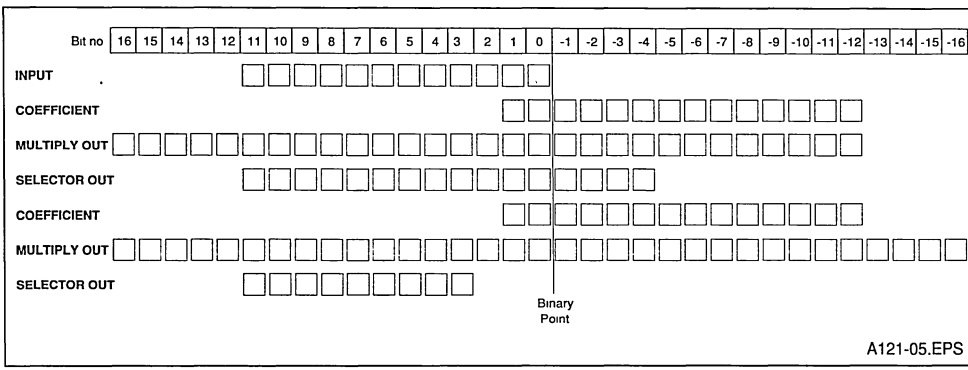
3. IDCT FUNCTION

The IDCT function is selected when **SEL[2-0]**=001, 101 or 111 (modes 1, 5 or 7).

3.1 Internal number format

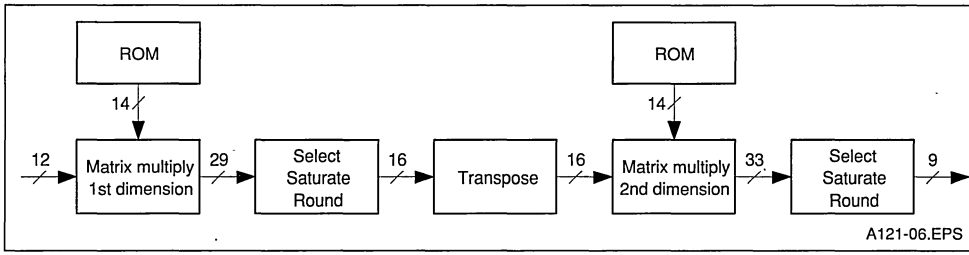
The input for the IDCT is a 12 bit signed integer in the range -2048 to +2047. The input is multiplied in the matrix multiplication array by 14 bit signed fixed point numbers in the range -2 to $2\text{-}2^{-12}$. The accumulated result of 8 multiply operations is a 29 bit signed integer, the bottom 8 bits of which are rounded (see section 1.3) and the top 5 bits used to saturate the output (see section 1.4). The result of the first matrix multiply is stored as a 16 bit signed integer and the second matrix multiply performed in exactly the same manner, yielding 33 bit results. The output rounds the bottom 19 bits, saturates the top 5 bits giving a 9 bit signed integer in the range -256 to +255

Figure 4 : IDCT Internal Number Format



3.2 Internal data flow

Figure 5 : Internal Data Flow



3.3 The mathematical basis for the IDCT

The 1 dimensional equation for the IDCT is as follows:

Inverse transform

$$x(m) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} X(k) c(k) \cos\left[\frac{(2m+1)k\pi}{2N}\right] \quad m=0,1,...,N-1$$

where

$$c(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } k=0 \\ 1 & \text{for } k=1...N-1 \end{cases}$$

where x(m) represent the output samples and X(k) is the input. The special case for the IMS A121 is for N=8 and the actual filter coefficients are then calculated. The following equation is used to calculate the actual filter coefficients.

IDCT coefficients

$$Coeff_{mk} = \sqrt{2}\, c(k) \cos\left[\frac{(2m+1)\, k\pi}{2N}\right]$$

It should be noticed that the coefficients are $2\sqrt{2}$ times bigger than in the inverse transform equation. This means hat the output after the 2 dimensional IDCT is 8 times too big (The 1 dimensional transform is applied twice giving $(2\sqrt{2})^2$ magnitude increase). This is in accordance with the 3 bit shift of the output data necessary to give the correct result.

IDCT coefficients

1.0000	1.3870	1.3066	1.1759	1.0000	0.7857	0.5412	0.2759
1.0000	1.1759	0.5412	-0.2759	-1.0000	-1.3870	-1.3066	-0.7857
1.0000	0.7857	-0.5412	-1.3870	-1.0000	0.2759	1.3066	1.1759
1.0000	0.2759	-1.3066	-0.7857	1.0000	1.1759	-0.5412	-1.3870
1.0000	-0.2759	-1.3066	0.7857	1.0000	-1.1759	-0.5412	1.3870
1.0000	-0.7857	-0.5412	1.3870	-1.0000	-0.2759	1.3066	-1.1759
1.0000	-1.1759	0.5412	0.2759	-1.0000	1.3870	-1.3066	0.7857
1.0000	-1.3870	1.3066	-1.1759	1.0000	-0.7857	0.5412	-0.2759

IDCT coefficients (14 bit signed integers)

4096	5681	5352	4816	4096	3218	2217	1130
4096	4816	2217	-1130	-4096	-5681	-5352	-3218
4096	3218	-2217	-5681	-4096	1130	5352	4816
4096	1130	-5352	-3218	4096	4816	-2217	-5681
4096	-1130	-5352	3218	4096	-4816	-2217	5681
4096	-3218	-2217	5681	-4096	-1130	5352	-4816
4096	-4816	2217	1130	-4096	5681	-5352	3218
4096	-5681	5352	-4816	4096	-3218	2217	-1130

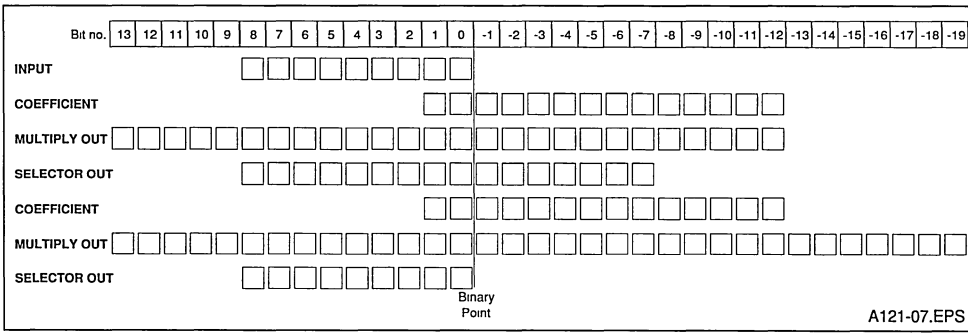
4. FILTER FUNCTION

The filter function is selected with **SEL[2-0]**=010. (mode 2)
This filter is intended to be used for image data, taking 9 bit signed input data and giving a 9 bit signed result.

4.1 Internal number format

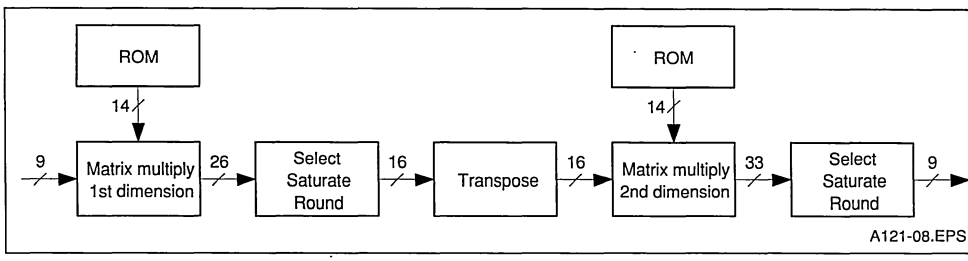
The input to the filter is a 9 bit signed integer in the range -256 to +255. The input is multiplied in the matrix multiplication array by 14 bit signed fixed-point numbers in the range -2 to $2\text{-}2^{-12}$. The accumulated result of 8 multiply operations is a 26 bit signed integer, the bottom 5 bits of which are rounded (see section 1.3) and the top 5 bits are used to saturate the output (see section 1.4). The result of the first matrix multiply is stored as a 16 bit signed integer and the second matrix multiply performed in exactly the same manner, yielding 33 bit results. The output rounds the bottom 19 bits, saturates the top 5 bits giving a 9 bit signed integer in the range -256 to +255.

Figure 6 : Internal Number Format



4.2 Internal data flow

Figure 7 : Internal Data Flow



4.3 Definition of filter

The filter is a simple $\frac{1}{4} - \frac{1}{2} - \frac{1}{4}$ filter applied in both dimensions which means that the overall filter kernel is:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

i.e. an output pixel is calculated from the corresponding pixel in the input field and its eight closest neighbours by evaluating

$$\frac{1}{16}(4\times \textit{pixel} + 2\times(\sum \textit{four adjacent pixels}) + 1\times(\sum \textit{four diagonal pixels}))$$

However, at the block edges, where some of the pixels would fall outside the block boundary, the filter is modified to 0--1--0 which means that along the edge the kernel would be:

$$\frac{1}{4}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (\textit{rotated suit})$$

and the corner pixels are passed through unmodified.

Filter coefficients

1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.2500	0.5000	0.2500	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.2500	0.5000	0.2500	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.2500	0.5000	0.2500	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.2500	0.5000	0.2500	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.2500	0.5000	0.2500	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.2500	0.5000	0.2500
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000

Filter coefficients (14 bit signed integers)

4096	0	0	0	0	0	0	0
1024	2048	1024	0	0	0	0	0
0	1024	2048	1024	0	0	0	0
0	0	1024	2048	1024	0	0	0
0	0	0	1024	2048	1024	0	0
0	0	0	0	1024	2048	1024	0
0	0	0	0	0	1024	2048	1024
0	0	0	0	0	0	0	4096

5. TRANSPOSER FUNCTION

The transposition function is selected with **SEL[2-0]**=011. (mode 3)
This is intended to be used for filtering image data, taking 9 bit signed input data and giving a 9 bit signed result. Data is passed through unmodified and is intended to be used in conjunction with the filter function (**SEL[2-0]**=010), so that by toggling **SEL[0]** the filter can be switched in and out.

5.1 Internal number format and data flow

The internal number format and data flow for the transpose function are the same as for the filter function. Refer to sections 4.1 and 4.2.

5.2 Transposition coefficients

1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000

5.3 Transposition coefficients (14 bit signed integers)

4096	0	0	0	0	0	0	0
0	4096	0	0	0	0	0	0
0	0	4096	0	0	0	0	0
0	0	0	4096	0	0	0	0
0	0	0	0	4096	0	0	0
0	0	0	0	0	4096	0	0
0	0	0	0	0	0	4096	0
0	0	0	0	0	0	0	4096

6. PIN DESIGNATIONS

System services

Pin	In/out	Function
V_{CC}, GND		Power supply and return
CLK	In	Input clock

Synchronous input/output

Pin	In/out	Function
GO	In	Initiate input/computation /output cycle
Din[11-0]	In	Data input port
Dout[11-0]	Out	Data output port
Dx[11-3]	In	Addition/subtraction port
SEL[2-0]	In	Mode select input port

6.1 System services

Power

Power is supplied to the device via the V_{CC} and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100nF low inductance (e.g. ceramic) capacitor between V_{CC} and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to V_{CC} and **GND**.

CLK

The clock input signal **CLK** controls the timing of input and the output on the three dedicated interfaces, and controls the progress of data through the addition/subtraction units, multipliers and transposition RAM. Since the IMS A121 is fully static, the clock can be stopped in either phase without corrupting data.

6.2 Synchronous input/output

GO

The **GO** signal is active high and is sampled on the rising edge of the input clock. If the device is processing a previous block of data, the **GO** signal is ignored. Otherwise, the processing of a block of 64 pixels commences and the **GO** signal is ignored for a further 63 cycles. Data is always assumed to be valid for the 64 cycles from the start of a major cycle. Blocks of data may be processed at any time and with any spacing between the major blocks, by toggling the **GO** signal as necessary.

Din[11-0]

The data input port is sampled 64 times on successive clock cycles, commencing when **GO** is sampled high. Data must be valid on the rising edge of **CLK** for each of the 64 cycles. The block of data may be considered as an 8×8 matrix, where each group of 8 samples represents a column, and the 8 columns are sampled consecutively until the block is complete. The data is twos complement, Little Endian so that **Din[11]** gives sign information, and **Din[12-x]** is the least significant bit where x is the word width indicated by the selected mode.

Dout[11-0]

The data output port will be valid for periods spanning 64 clock cycles. The data will be valid on the rising edge of the clock, exactly 128 cycles (the latency) after the data was sampled on the input. This output data, which may be considered as an 8×8 matrix, is transposed with respect to the input data. The data is twos complement, Little Endian like the input data.

Blocks of data may follow directly after one-another so that the first data of a block is presented exactly 64 cycles after the first data of the preceding block. However, if there is a gap between blocks zero will appear on the data output port between blocks of data.

Dx[11-3]

The addition/subtraction port is sampled on each clock cycle in exactly the same way as the data input port. The data on this port will either be subtracted from the signal on the data input port before matrix multiplication, or, added to the result of matrix multiplication prior to output. The addition and subtraction functions can never be used together. The function selected is determined by the **SEL[2-0]** signals. The data is twos complement, Little Endian like the **Din/Dout** data. Note however, that although the **Dx** port has a different width, **Dx[10]** has the same bitwise significance as **Din[10]/Dout[10]**.

The timing of data on the **Dx** port is different depending on the selected mode.

In the case of subtraction in the DCT mode, **SEL[1-0]=00**, data is presented on the **Dx** port on the same cycle as the corresponding data (from which it will be subtracted) is presented on the **Din** port.

In the case of addition in the IDCT mode, **SEL[1-0]=01**, data is presented on the **Dx** port exactly 4 cycles before the corresponding data (to which it will have been added) appears on the **Dout** port.

SEL[2-0]

The mode select input port is sampled on the rising edge of **CLK**, when **GO** is active, at the start of a block of data. This fixes the selected mode for the entire block of data.

SEL[2-0]	Mode	Function	PreSubtract	PostAdd	Clipping	Din width	Dout width
000	0	DCT	Disabled	Disabled	Disabled	9	12
001	1	IDCT	Disabled	Disabled	Disabled	12	9
010	2	Filter	Disabled	Disabled	Disabled	9	9
011	3	Transpose	Disabled	Disabled	Disabled	9	9
100	4	DCT	Enabled	Disabled	Disabled	9	12
101	5	IDCT	Disabled	Enabled	Enabled	12	9
110	6	Reserved–Do not use					
111	7	IDCT	Disabled	Disabled	Enabled	12	9

7. ELECTRICAL SPECIFICATION

7.1 DC electrical characteristics

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Min	Typ.	Max	Units	Notes (1)
V_{CC}	DC supply voltage	0		7.0	V	2
V_I, V_O	Voltage on input and output pins	-1.0		V_{CC}+0.5	V	2
T_A	Temperature under bias	-40		85	°C	2
T_{stg}	Storage temperature	-65		150	°C	2
P_{Dmax}	Power dissipation			1.5	W	2

Notes

1 All voltages are with respect to GND.

2 This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

DC OPERATING CONDITIONS

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes (1)
V_{CC}	DC supply Voltage	4.5	5.0	5.5	V	
V_{IH}	Input Logic '1' Voltage	2.0		V_{CC}+0.5	V	2
V_{IL}	Input Logic '0' Voltage	-0.5		0.8	V	2
T_A	Ambient Operating Temperature	0		70	°C	3

Notes

1 All voltages are with respect to GND. All **GND** pins must be connected to GND.

2 nput signal transients up to 10 ns wide, are permitted in the voltage ranges GND - 0.5 V to GND - 1.0 V and V_{CC} + 0.5 V to V_{CC} + 1.0 V.

3 400 linear ft/min transverse air flow.

DC CHARACTERISTICS

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes (1,2)
V_{OH}	Output Logic '1' Voltage	2.4		V_{CC}	V	$I_O \leq -4.4mA$
V_{OL}	Output Logic '0' Voltage	0		0.4	V	$I_O \leq 4.4mA$
I_{IN}	Input Leakage Current (any input)			±10	µA	3
I_{CC}	Average Power Supply Current			300	mA	4

Notes

1 All voltages are with respect to GND. All **GND** pins must be connected to GND.
2 Under the conditions specified by the DC operating conditions.
3 $V_{CC} = V_{CC}(max)$, $GND \leq V_{IN} \leq V_{CC}$
4 This applies at 20 MHz and will be less at slower clock rates

7.2 A.C. timing characteristics

All timings are given for a load of 30pF unless otherwise stated.

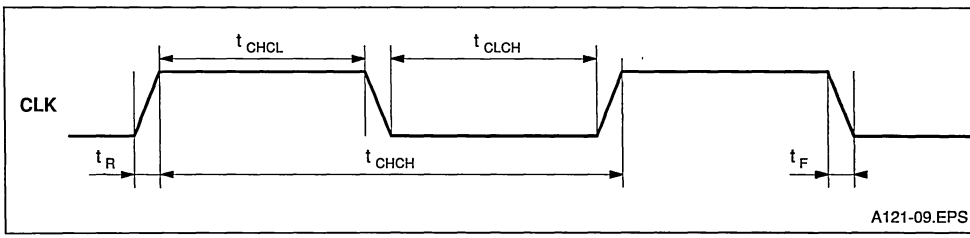
CLOCK REQUIREMENTS

Symbol	Parameter	Min	Typ.	Max	Units	Notes
t_{CHCL}	Clock Pulse High Width	20			ns	
t_{CLCH}	Clock Pulse Low Width	20			ns	
t_{CHCH}	Clock Period	50			ns	
t_R	Clock Rise Time	0		50	ns	1
t_F	Clock Fall Time	0		50	ns	1

Notes

1 The clock edges should be monotonic between V_{IL} and V_{IH}.

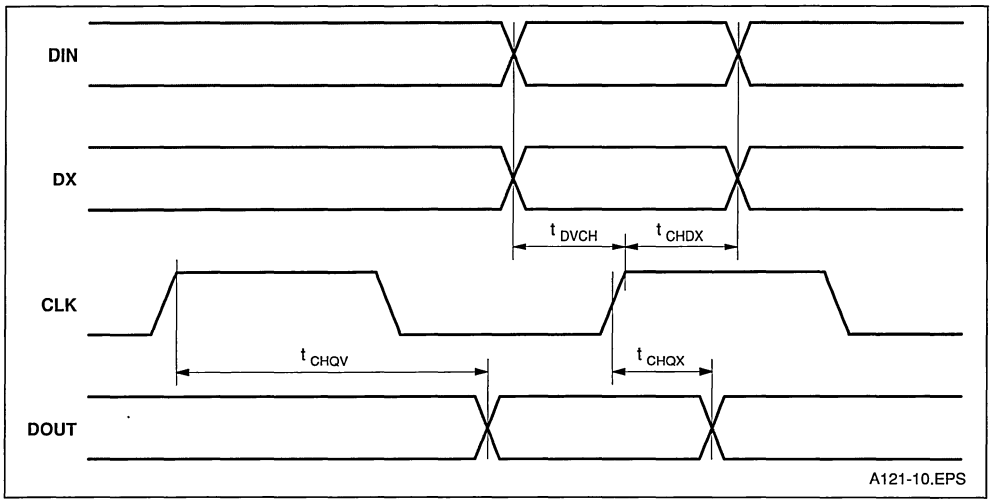
Figure 8



SYNCHRONOUS INPUT AND OUTPUT (Din, Dout, Dx)

Symbol	Parameter	Min	Typ.	Max	Units	Notes
t_{CHQV}	CLK High to Dout Valid			38	ns	
t_{CHQX}	Dout Hold Time after CLK	2			ns	
t_{DVCH}	Din/Dx Setup Time to CLK High	10			ns	
t_{CHDX}	Din/Dx Hold Time to CLK High	0			ns	

Figure 9



Synchronous control (GO, SEL[2-0])

Symbol	Parameter	Min	Typ.	Max	Units	Notes
t_{GHCH}	GO/SEL hold to clock high	0			ns	
t_{GSCH}	GO/SEL setup to clock high	10			ns	

Figure 10

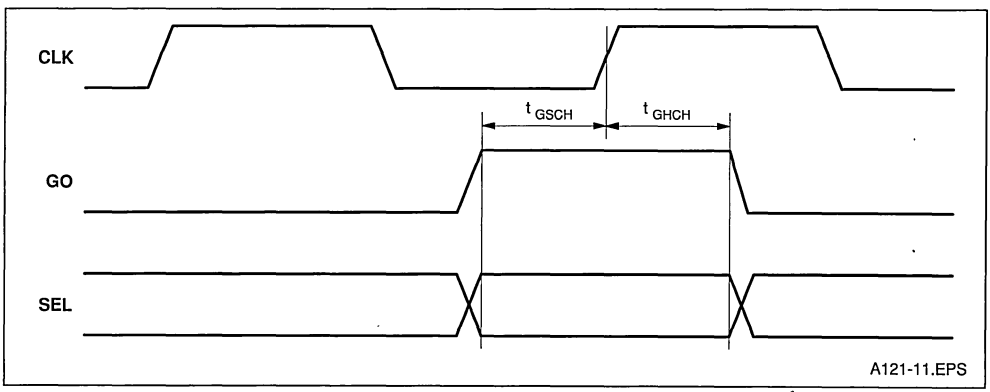
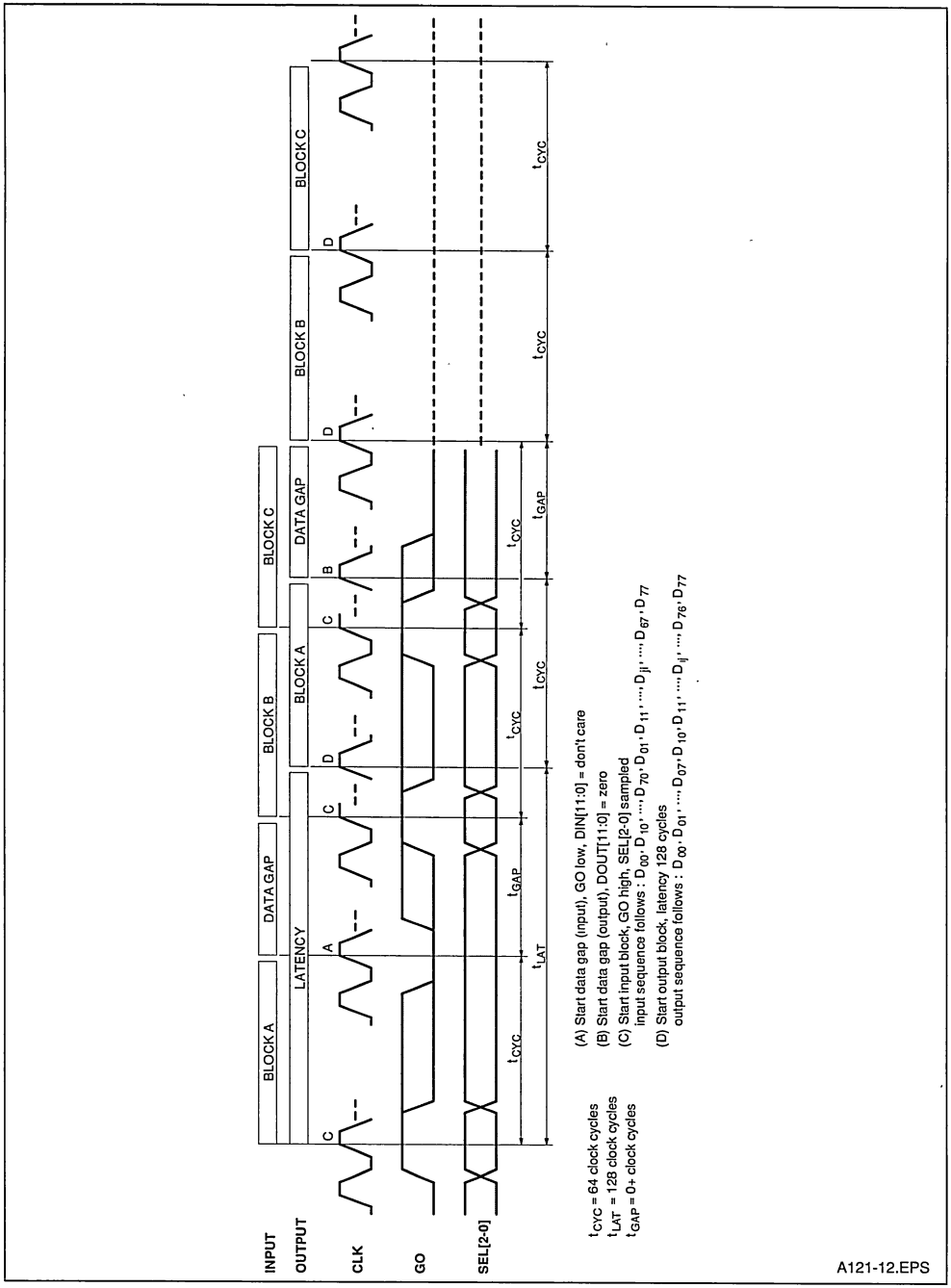
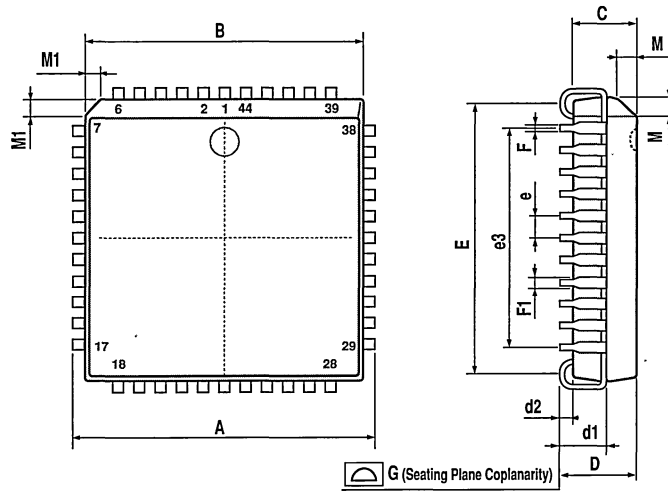


Figure 11 : Overall Data Timing



PACKAGE MECHANICAL DATA

44 PINS - PLASTIC CHIP CARRIER



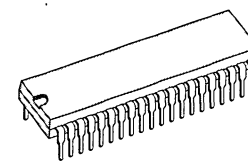
Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A		17.577			0.692	
B		16.612			0.654	
C		3.861			0.152	
D		4.369			0.172	
d1						
d2						
E						
e		1.270			0.050	
e3		12.70			0.500	
F						
F1		0.457			0.018	
G						
M						
M1		1.143			0.045	

PLCC44B.TBL

DISCRETE COSINE TRANSFORM (DCT)

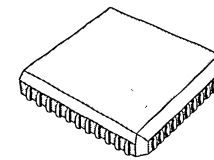
- 0 TO 15.0 MHz OPERATING FREQUENCY EQUAL TO PIXEL RATE
- FORWARD OR INVERSE TRANSFORM
- 7 BLOCK SIZE POSSIBILITIES :

  16 x 16 8 x 8 4 x 4
  16 x 8 8 x 4
  8 x 16 4 x 8
- 9-BIT TWO'S COMPLEMENT PIXEL FORMAT CORRESPONDING TO 3 POSSIBLE MAGNITUDES DEPENDING ON THE PIXEL RANGE PIN (PR) STATE :
  - 8-BIT UNSIGNED MAGNITUDE
  - 8-BIT 2's COMPLEMENT MAGNITUDE
  - 9-BIT 2's COMPLEMENT MAGNITUDE
- 12-BIT TWO'S COMPLEMENT COEFFICIENT FORMAT
- FULLY TTL AND CMOS COMPATIBLE
- CMOS TECHNOLOGY
- SINGLE + 5 VOLTS POWER SUPPLY
- POWER DISSIPATION : 500 mW AT 15.0 MHz



DIP40
(Plastic Package)

ORDER CODE : STV3200P



PLCC44
(Plastic Chip Carrier)

ORDER CODE : STV3200FN

DESCRIPTION

The STV3200 is a dedicated circuit for the discrete cosine transform (DCT) computation. The two-dimensional forward DCT (FDCT) or inverse DCT (IDCT) is performed for various block sizes and a pixel rate up to 15.0 MHz. The circuit architecture is fully bidirectional with a 9-bit magnitude pixel data bus and a 12-bit magnitude coefficient data bus programmed as input or output depending on the selection of FDCT or IDCT.

	FDCT	IDCT	Data Format
Pixel Bus	Input	Output	9-bit 2's Complement
Coefficient Bus	Output	Input	12-bit 2's Complement

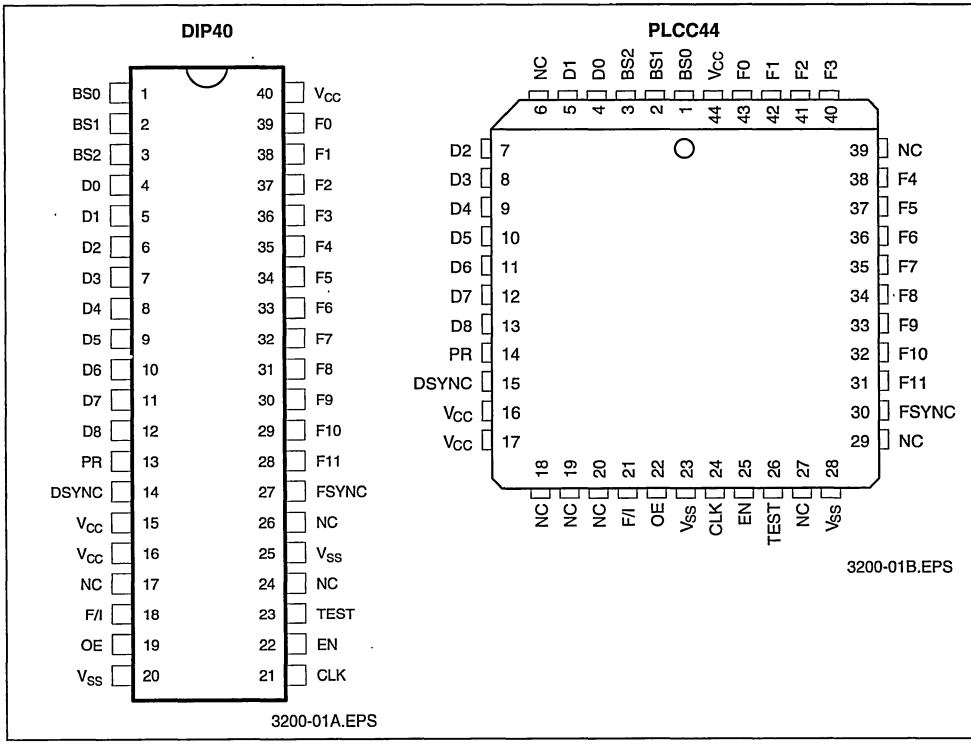
For the forward transform, the input pixels are coded in 9-bit 2's complement and the output coefficients are coded in 12-bit 2's complement. For the inverse transform, the data format is identical with the coefficients used as input and the pixels used as output.

ORDER CODES

Part Number	Temperature Range	Package
STV3200CP	0 to 70°C	DIP 40
STV3200CFN	0 to 70°C	PLCC44

3200-01.TBL

PIN CONNECTIONS

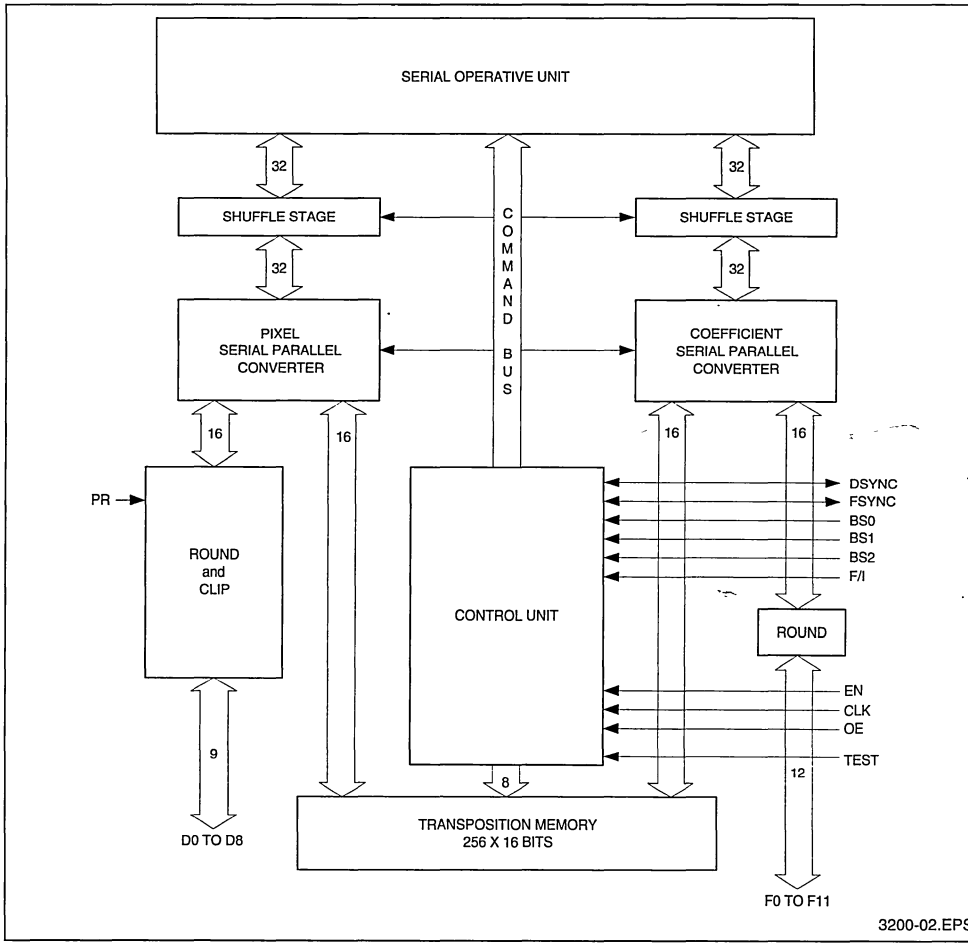


PIN DESCRIPTION

DIP40	PLCC44	Symbol	Direction	Function
1 : 3	1 : 3	BS0 to BS2	IN	Block Size Selection
4 : 12	4 : 5 , 7 : 13	D0 to D8	IN / OUT	Pixel Data Bus
13	14	PR	IN	Pixel Range Selection
14	15	DSYNC	IN / OUT	Pixel Block Synchronization
18	21	F/I	IN	Forward or Inverse Transform Selection
19	22	OE	IN	Tristate Output Control
20 - 25	23 - 28	V_{SS}		Ground
21	24	CLK	IN	Clock Input
22	25	EN	IN	Clock Enable
23	26	TEST	IN	Test Mode
27	30	FSYNC	IN / OUT	Coefficient Block Synchronization
28 : 39	43 : 40 , 38 : 31	F0 to F11	IN / OUT	Coefficient Data Bus
40 - 15 - 16	16 - 17 - 44	V_{CC}		+ 5V ± 10%
17 - 24 - 26	6 - 18 - 19 - 20 27 - 29 - 39	NC		Not Connected

3200-02.TBL

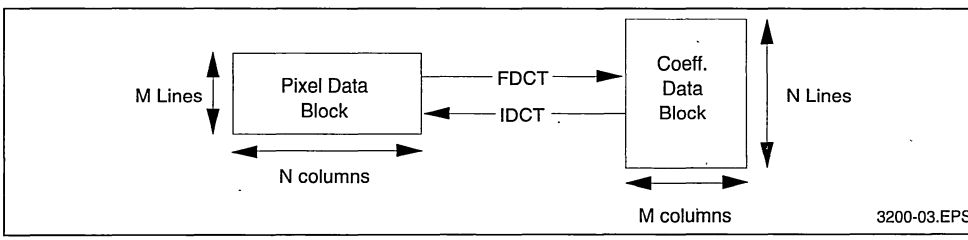
FUNCTIONAL BLOCK DIAGRAM



FUNCTIONAL DESCRIPTION

1. EQUATIONS

Figure 1



The STV3200 performs a Two dimensional Discrete Cosine Transform according to the following equations, where the block size is defined by M lines and N columns of pixels :

FORWARD TRANSFORM EQUATION :

$$F(u,v) = \text{Round}\left[\frac{32}{N \cdot M} C^2(u)\, C^2(v) \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} D(i,j) \cos\frac{(2 \cdot i + 1)\, u\, \pi}{2 \cdot M} \cos\frac{(2 \cdot j + 1)\, v\, \pi}{2 \cdot N}\right]$$

where C2(u) = 1/2 if u = 0
= 1 otherwise

INVERSE TRANSFORM EQUATION :

$$D(i,j) = \text{Round}\left[\frac{1}{8} \sum_{V=0}^{N-1} \sum_{u=0}^{M-1} F(u,v) \cos\frac{(2 \cdot i + 1)\, u\, \pi}{2 \cdot M} \cos\frac{(2 \cdot j + 1)\, v\, \pi}{2 \cdot N}\right]$$

2. BLOCK FORMAT

MÃN is the block size. This means that pixel blocks contain N columns of M pixels. The STV3200 performs a block transposition, and therefore the coefficient blocks contain M columns of N pixels.

The seven different possible block sizes are : 16 x 16, 8 x 16, 16 x 8, 8 x 8, 4 x 8, 8 x 4 and 4 x 4.

3. BLOCK SCANNING

Many possible arrangements for pixel block scanning are possible. These different arrangements are :

A - the block is entered line by line from the top line to the bottom line. Each line is entered from the left pixel to the right pixel.

B - the block is entered line by line from the top line to the bottom line. Each line is entered from the right pixel to the left pixel.

C - the block is entered line by line from the bottom line to the top line. Each line is entered from the left pixel to the right pixel.

D - the block is entered line by line from the bottom line to the top line. Each line is entered from the right pixel to the left pixel.

E - the block is entered column by column from the left column to the right column. Each column is entered from the top pixel to the bottom pixel.

F - the block is entered column by column from the left column to the right column. Each column is entered from the bottom pixel to the top pixel.

G - the block is entered column by column from the right column to the left column. Each column is entered from the top pixel to the bottom pixel.

H - the block is entered column by column from the right column to the left column. Each column is entered from the bottom pixel to the top pixel.

4. DATA FORMAT

The coefficient format is 12-bit 2's Complement, corresponding to the range – 2048 to 2047.

There are 3 possible ranges for pixel data :

8-BIT UNSIGNED PIXEL MAGNITUDE
(see Figure 2)
The pixel data range is 0 to 255. In this case D8 is always equal to 0 and the PR pin must be set to 1 for FDCT and IDCT. A clipping to the range 0 to 255 is performed before outputting the reconstructed pixels after an IDCT.

8-BIT TWO'S COMPLEMENT MAGNITUDE
(see Figure 3)
The input pixel data range is – 256 to 255 and a clipping to the range – 128 to 127 is performed internally just before the FDCT. In this case, the PR pin must be set to 0 for FDCT and IDCT.

9-BIT TWO'S COMPLEMENT MAGNITUDE
(see Figure 4)
The input pixel data range is – 256 to 255 and no clipping is performed. In this case, the PR pin must be set to 1 for FDCT and 0 for IDCT. Internal overflows may occur leading to aberrant errors on reconstructed values.

Figure 2

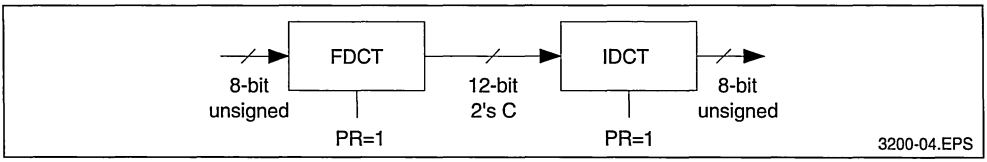


Figure 3

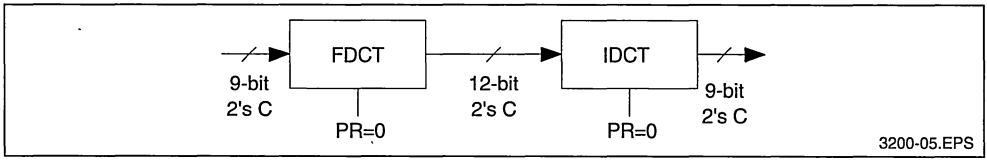
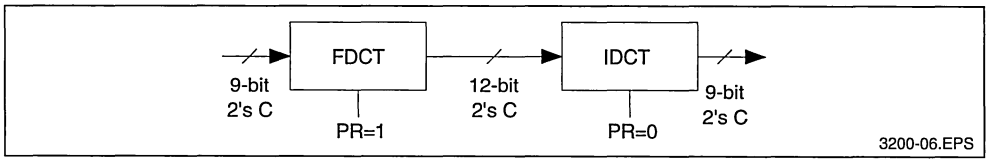


Figure 4



5. BLOCK FLOW

Depending on the application, blocks may be entered in different ways.

Latent period : the latent period between input data and the corresponding output results in 130 + M·N cycles. This means that the first data item of a resulting block is provided 130 + M·N clock cycles after the first data item of the corresponding input block.

Synchronization signals : an input block synchronization signal must be provided. The input pin for this signal is DSYNC if FDCT is selected and FSYNC if IDCT is selected. This signal must be active with the first data item of each input block or group of blocks.

An output block synchronization signal is provided. The output pin for this signal is FSYNC if FDCT is selected and DSYNC if IDCT is selected. This signal is active with the first data item of each output block or group of blocks.

The output synchronization signal is equal to the input synchronization signal delayed from 130 + M·N clock cycles.

CONTINUOUS BLOCK FLOW

Input data is entered continuously with one new item data at each clock cycle and output data is provided continuously with one new result data item at each clock cycle.

The input synchronization pulse can be provided for each input block. In this case the output synchronization pulse is provided for each output block (Figure 5). Another way is to provide a synchronization pulse only for the first block. In this case, only one synchronization pulse is provided for the first output block (Figure 6).

Figure 5 : Continuous Block Flow-1

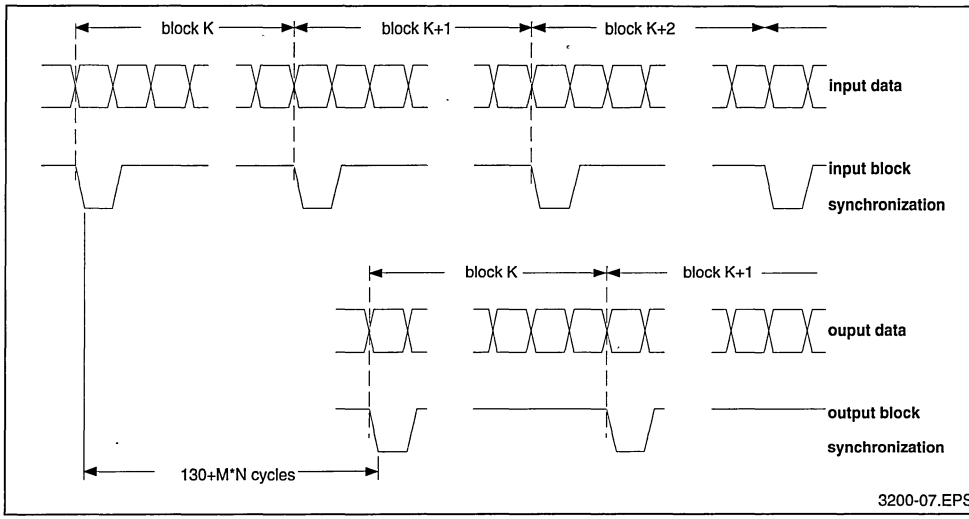
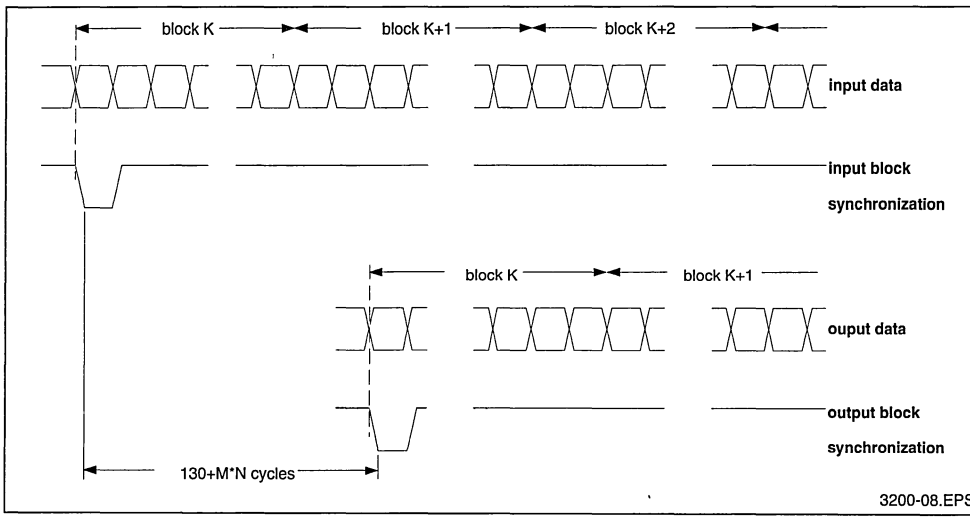


Figure 6 : Continuous Block Flow-2



CONTINUOUS BLOCK FLOW WITH BYPASS OF IRRELEVANT DATA

It is possible to process a block flow including irrelevant data (corresponding to line suppresssion for example) as if it was a continuous block flow. One way is to stop the clock signal during the irrelevant data occurrence. Another way is to use the Clock Enable Signal (EN) to inhibit the chip internal clock during irrelevant data occurrence (Figure 7).

BURST BLOCK FLOW (see Figure 8)
Single blocks (or groups of blocks) may not be contiguous. In other words, delay cycles between two blocks (or groups of blocks) may exist. During these delay cycles, the clock is still running and the chip continues to perform computations. The constraint is that the internal pipe line must not be broken when a new block occurs. To take this constraint into account, the number of delay cycles (NC) must respect one of the following conditions :

1 - the number of delay cycles (NC) is greater than or equal to 130 + M·N. In this case the pipe line is empty (all the relevant data has been outputted) when a new input block processing starts.

2 - the number of delay cycles (NC) is a multiple of the number of cycles required to enter a block (M·N). In this case, the input data always remains synchronous with the internal pipe line.

MIXED FDCT/IDCT (see Figure 9)
In some low frequency applications, it could be useful to use only one chip to compute all the FDCT and IDCT required by the coding scheme. Blocks must be entered in a burst fashion with at least 130 + M·N delay cycles between the last item of the set of input pixels for FDCT and the first item of input coefficients for IDCT. The same delay must be respected between the last item of input coefficients for IDCT and the first pixel of input pixels for FDCT.

Figure 7 : Continuous Block Flow with Irrelevant Data

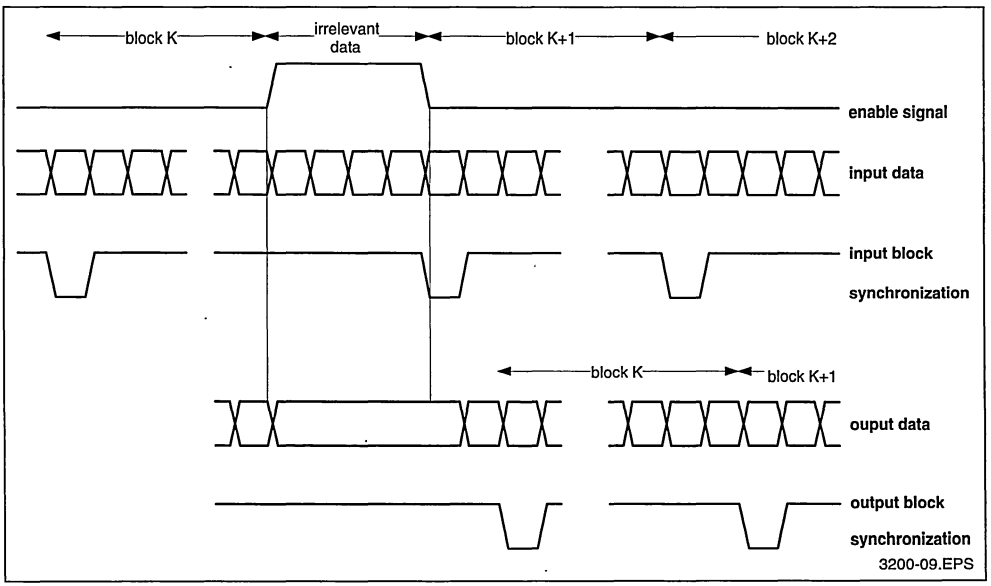


Figure 8 : Burst Block Flow

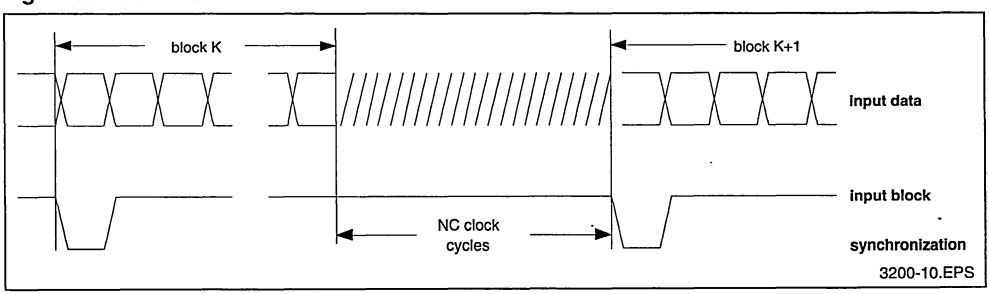
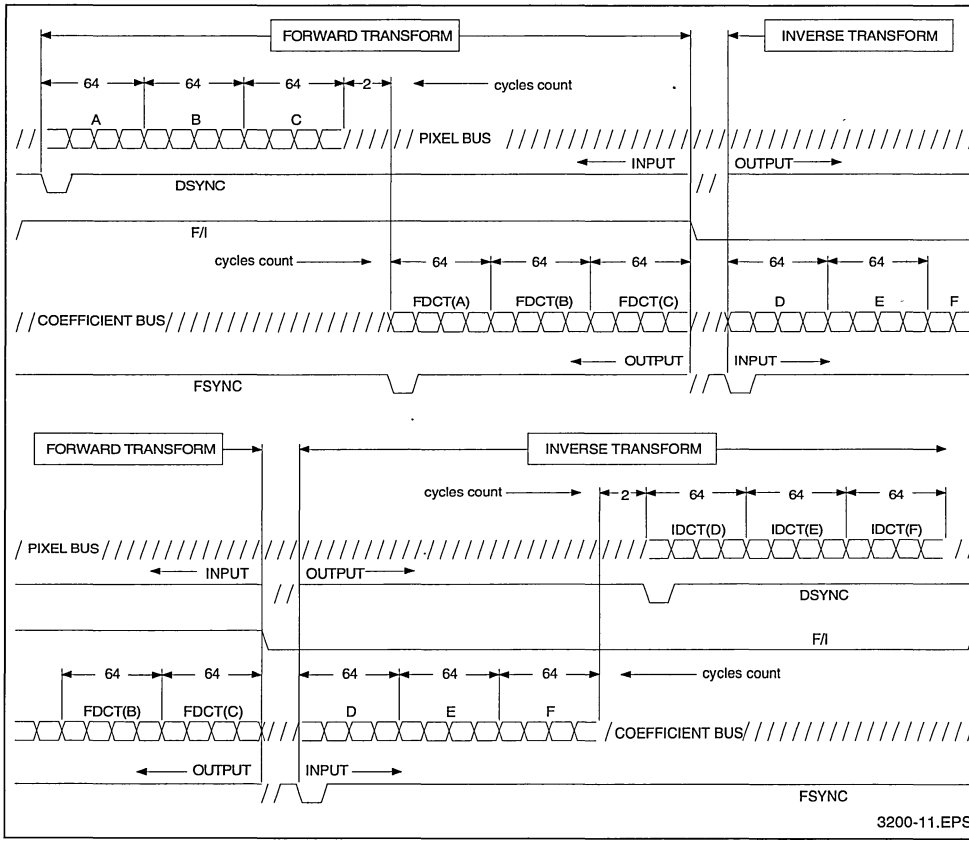


Figure 9 : Mixed 8x8 FDCT/IDCT Example Waveforms



6. PINS DESCRIPTION

CLK : Clock signal

DATA PINS

D0 TO D8 : 9-bit bidirectional Pixel data bus pins. Direction is programmed by the F/I pin :

F/I State	D0 to D8 Direction
High	Input
Low	Output

Data is loaded (when input) on the falling edge of CLK or settled (when output) on the rising edge of CLK. D0 is the least significant bit and D8 the most significant one.

MSB								LSB	
D8	D7	D6	D5	D4	D3	D2	D1	D0	Pin
-256	128	64	32	16	8	4	2	1	Weight

DSYNC : Pixel data block synchronization signal. This pin is bidirectional with the direction programmed by the F/I pin (like D0 to D8). DSYNC is active (low level) with the first pixel data of a block (or group of blocks).

F0 TO F11 : 12-bit bidirectional Coefficient data bus pins. Direction is programmed by the F/I pin :

F/I State	F0 to F11 Direction
High	Input
Low	Output

Data is loaded (when input) on the falling edge of CLK or settled (when output) on the rising edge of CLK. F0 is the least significant bit and F11 the most significant one.

MSB											LSB	
F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0	Pin
-2048	1024	512	256	128	64	32	16	8	4	2	1	Weight

FSYNC : Coefficient data block synchronization signal. This pin is bidirectional with the direction programmed by the F/I pin like F0 to F11. FSYNC is active (low level) with the first coefficient data of block (or group of blocks).

CONTROL PINS

F/I : Forward or inverse selection. When F/I is high, forward DCT is performed. When F/I is low, inverse DCT is performed.

BS0 to BS2 : Block size selection. The block size is programmed through these three pins according to the following table :

BS0	BS1	BS2	Pixel Block Size	Coefficient Block Size
0	0	0	16 * 16	16 * 16
0	0	1	8 * 16	16 * 8
0	1	0	16 * 8	8 * 16
0	1	1	8 * 8	8 * 8
1	0	0	4 * 8	8 * 4
1	0	1	8 * 4	4 * 8
1	1	0	4 * 4	4 * 4
1	1	1	Reserved	

PR : Pixel range selection. This pin controls the clipping of the pixel data. If PR is high, output pixels of an IDCT are clipped to the range 0 to 255. If PR is low input pixels of an FDCT are clipped to the range – 128 to 127.

OE : Output enable. This signal is active low. When OE is high, all outputs (defined by the F/I pin state) are forced to the high impedance state.

EN : Enable. This signal is active low. When EN is high, internal states of the chip are frozen. When EN becomes low, execution restarts.

POWER SUPPLY AND GROUND PINS

V_{CC} : + 5 Volt power supply

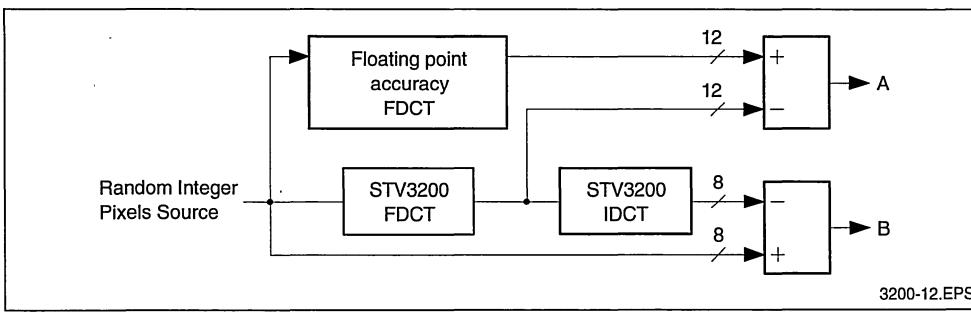
V_{SS} : ground

OTHERS

TEST : test control. This pin is reserved and must be low in normal mode.

7. ACCURACY CHARACTERISTICS

The accuracy characteristics have been measured according to the following scheme :



A : characteristics of FDCT for 8-bit magnitude random pixel data. Error between the FDCT computed with floating point accuracy and the FDCT computed by the STV3200 is measured.

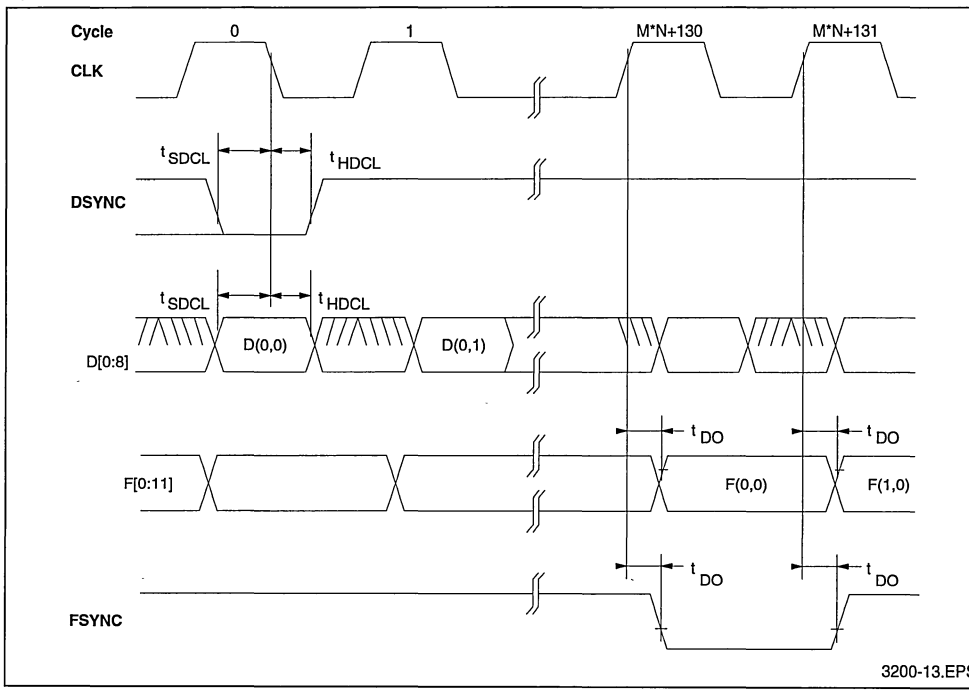
Block Size	16 * 16	8 * 16	16 * 8	8 * 8	4 * 8	8 * 4	4 * 4
Exact Value	86.8%	88.4%	86.9%	88.4%	89.4%	8.8%	91.0%
Error of ± 1LSB	13.0%	11.4%	13.0%	11.5%	10.5%	11.1%	8.9%
Error of ± 2LSB	0.23%	0.19%	0.14%	0.11%	0.11%	0.07%	0.11%

B : characteristics of FDCT followed by an IDCT for 8-bit magnitude random pixel data. Error between the source picture and the FDCT computed by the STV3200 followed by an IDCT computed by the STV3200 is measured.

Block Size	16 * 16	8 * 16	16 * 8	8 * 8	4 * 8	8 * 4	4 * 4
Exact Value	82.2%	93.9%	93.0%	99.0%	99.9%	99.9%	100%
Error of ± 1LSB	17.8%	6.1%	7.0%	1.0%	0.06%	0.08%	0%
Error of ± 2LSB	0.02%	0%	0%	0%	0%	0%	0%

TIMING WAVEFORMS

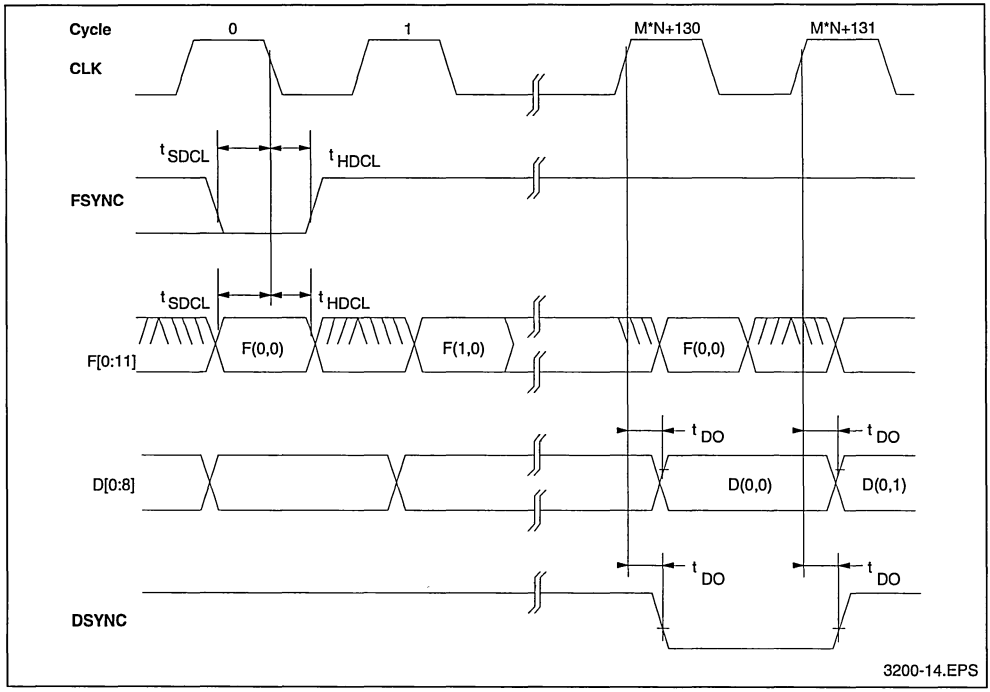
Sync Signals Timing Diagram for a Forward Transform on a M*N Block.



Note : FSYNC will be in an unknown state during the first 130 cycles after the power-up.

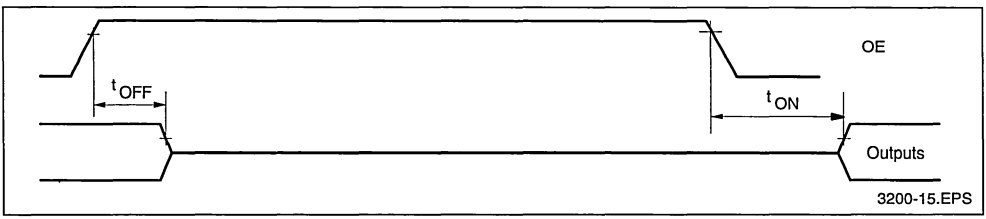
TIMING WAVEFORMS (continued)

Sync Signals Timing Diagram for a Inverse Transform on a M*N Block.

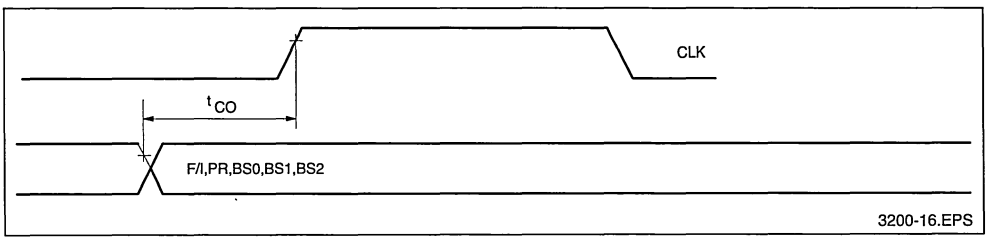


Note : DSYNC will be in an unknown state during the first 130 cycles after the power-up.

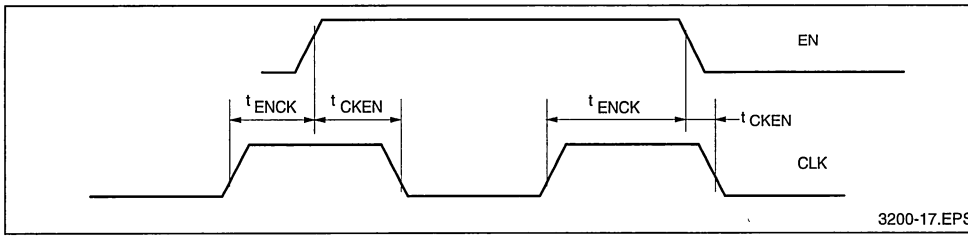
Outputs Enable Signal Timing Diagram



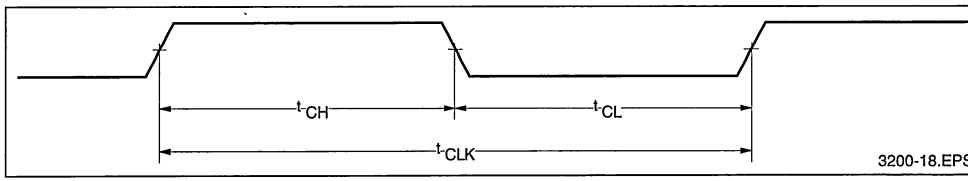
Control Signal Timing Diagram



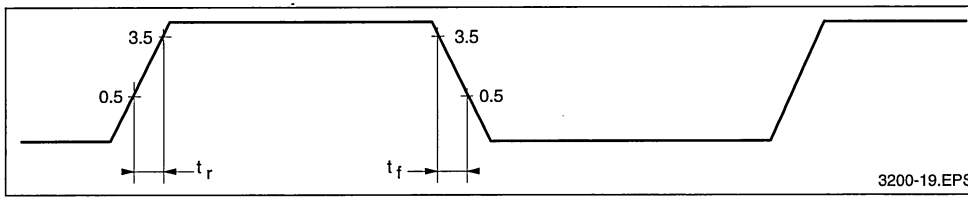
Enable Signal Timing Diagram



Clock Timing Diagram



Output Timing Diagram



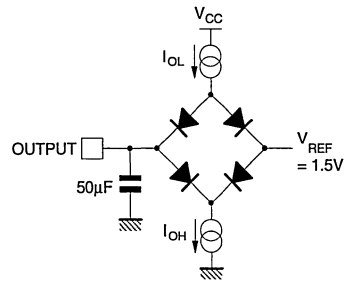
ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS
Supply voltage (V_{CC}) : 6 Volts
Operating Temperature Range : 0 to 70 °C
Voltage on Any Pin Relative to V_{SS} : − 0.5 to V_{CC} + 0.5 Volts

DC ELECTRICAL CHARACTERISTICS
Operating conditions : V_{SS} = 0 Volt, T_A = 0 to 70 °C, V_{CC} = 5V ± 10 % unless otherwise noted

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{CC}	Operating Voltage		4.5		5.5	V
	Power Supply Ripple				0.5	V
I_{CC}	Supply Current f_{CLK} = 15MHz f_{CLK} = 0MHz	C_{LOAD} = 50pF on all output All inputs at V_{CC} or V_{SS}			 100 1	 mA mA
V_{IL} V_{IH}	Input Voltage Level (all inputs) Logic Low Logic High Hi-Z Input Leakage IN/OUT Buffers Input Buffers	V_{CC} = 5 ± 0.5 V_{IN} = V_{SS} to V_{CC}	 2 -5 -1		 0.8 +5 +1	 V V mA mA
V_{OL} V_{OH}	Output Voltage Level (all outputs) Logic Low, ILOAD = +500µA Logic High, ILOAD = -500µA	V_{CC} = 4.5V	 2.7		 0.4	 V V
C_{IN}	Input Capacitance	V_{offset} = 2.5V, f = 1MHz			10	pF

AC ELECTRICAL CHARACTERISTICS
Operating Conditions : V_{SS} = 0 Volt, T_A = 0 to 70 °C, V_{CC} = 5 V ± 10 % unless otherwise noted
Outputs Load : capacitance = 50pF, current logic low = 500µA
Test Load on All Outputs :



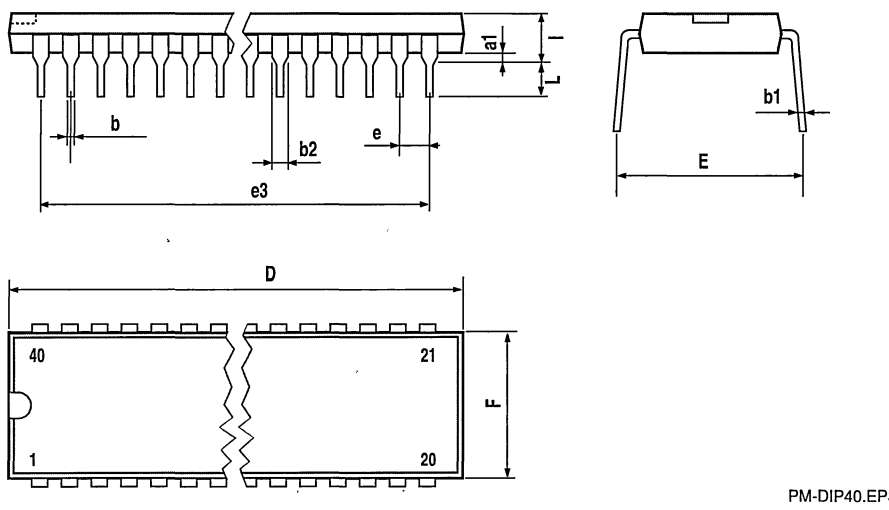
3200-20.EPS

Timings are measured between threshold voltage of 1.5 V unless otherwise specified.

Symbol	Parameter	Min.	Typ.	Max.	Unit
t_R	Rise Time from 0.5 to 3.5V			10	ns
t_F	Fall Time from 3.5 to 0.5V			10	ns
t_{CH}	Clock High Pulse Width	30			ns
t_{CL}	Clock Low Pulse Width	30			ns
t_{CLK}	Clock Cycle	66			ns
t_{SDCL}	Data Setup Time from CLK	5			ns
t_{HDCL}	Data Hold Time from CLK	20			ns
t_{DO}	Output Data Delay from CLK			33	ns
t_{CKEN}	Enable Hold from CLK	5			ns
t_{ENCK}	Enable Setup from CLK	5			ns
t_{OFF}	Delay from OE ↑ to Output going to High Impedance State			20	ns
t_{ON}	Delay from OE ↓ to Output going to High or Low State			20	ns
t_{CO}	Control Signal Setup from beginning of Input Stream	100			ns

PACKAGE MECHANICAL DATA

40 PINS - PLASTIC DIP

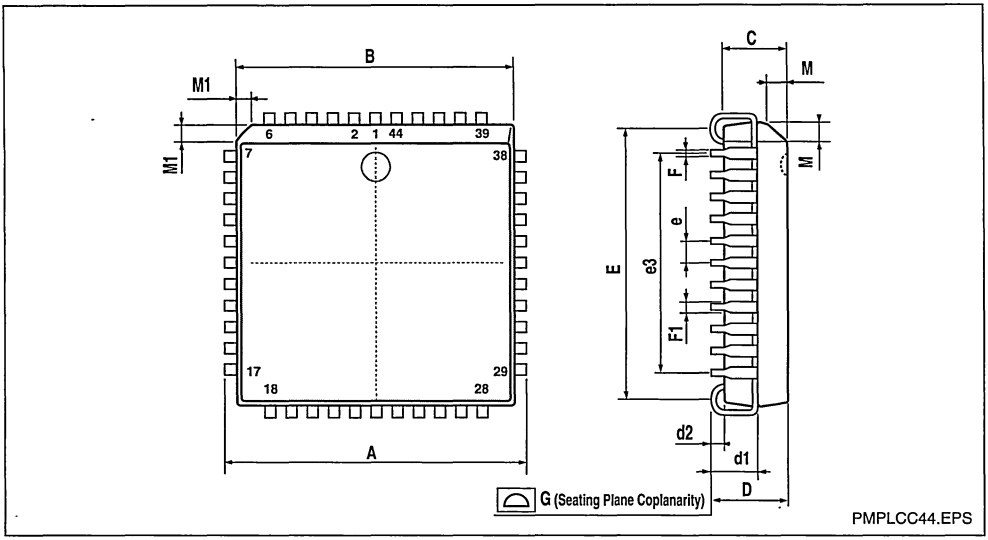


Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
a1		0.63			0.025	
b		0.45			0.018	
b1	0.23		0.31	0.009		0.012
b2		1.27			0.050	
D			52.58			2.070
E	15.2		16.68	0.598		0.657
e		2.54			0.100	
e3		48.26			1.900	
F			14.1			0.555
i		4.445			0.175	
L		3.3			0.130	

DIP40.TBL

PACKAGE MECHANICAL DATA

44 PINS - PLASTIC CHIP CARRIER



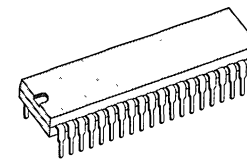
Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A	17.4		17.65	0.685		0.695
B	16.51		16.65	0.650		0.656
C	3.65		3.7	0.144		0.146
D	4.2		4.57	0.165		0.180
d1	2.59		2.74	0.102		0.108
d2		0.68			0.027	
E	14.99		16	0.590		0.630
e		1.27			0.050	
e3		12.7			0.500	
F		0.46			0.018	
F1		0.71			0.028	
G			0.101			0.004
M		1.16			0.046	
M1		1.14			0.045	

PLCC44.TBL

8 x 8 DISCRETE COSINE TRANSFORM (DCT)

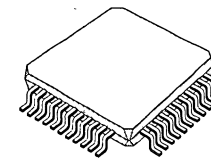
ADVANCE DATA

- 0 TO 27MHz PIXEL RATE IN SINGLE PRECISION MODE,
- 0 TO 20 MHz PIXEL RATE IN DOUBLE PRECISION MODE
- FORWARD AND INVERSE 8 x 8 TRANSFORM
- 9-BIT TWO'S COMPLEMENT PIXEL FORMAT
- 12-BIT TWO'S COMPLEMENT COEFFICIENT FORMAT
- OPTIMIZED ACCURACY FOR 8-BIT TWO'S COMPLEMENT PIXEL FORMAT
- SELECTABLE SCANNING OF COEFFICIENT BLOCKS
- FULLY TTL AND CMOS COMPATIBLE
- CMOS TECHNOLOGY
- SINGLE +5 VOLT POWER SUPPLY
- MAXIMUM POWER DISSIPATION : 750mW AT 27MHz



DIP40
(Plastic Package)

ORDER CODE : STV3208CP



PQFP44
(Plastic Package)

ORDER CODE : STV3208CV

DESCRIPTION

The STV3208 is a dedicated circuit for the 8 x 8 discrete cosine transform (DCT) computation. Two-dimensional forward DCT (FDCT) or inverse DCT (IDCT) is performed for 8 x 8 block sizes and a pixel rate up to 27MHz. The circuit architecture is fully bidirectional with 9-bit magnitude pixel data bus and a 12-bit magnitude coefficient data bus programmed as input or output depending on the selection of FDCT or IDCT.

	FDCT	IDCT	Data Format
Pixel Bus	Input	Output	9-bit 2's Complement
Coefficient Bus	Output	Input	12-bit 2's Complement

For the forward transform, the input pixels are coded on 9-bit 2's complement and the output coefficients are coded on 12-bit 2's complement. For the inverse transform, the data format is identical with the coefficients used as input and the pixels used as output.

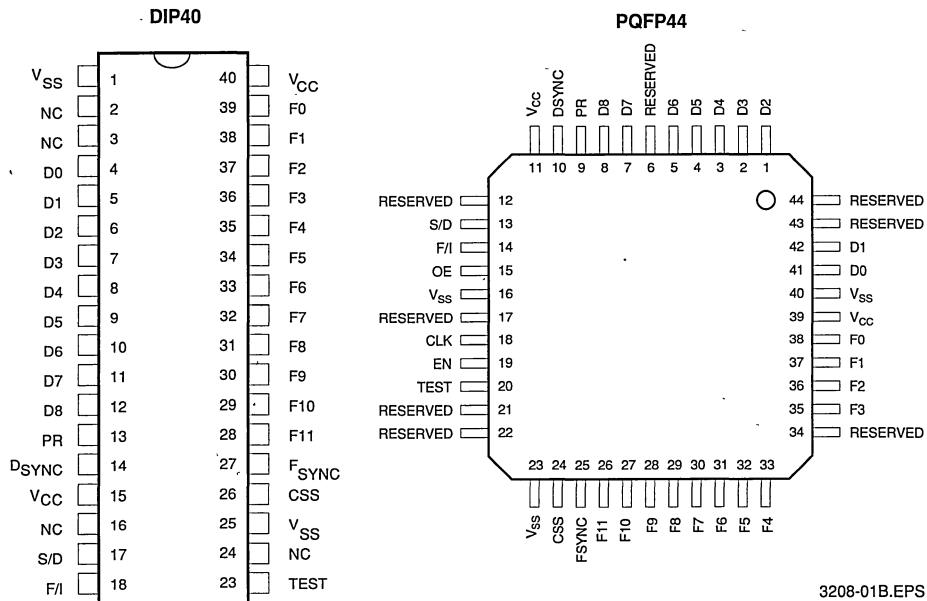
Two operating modes are provided : single precision mode at a pixel rate up to 27 MHz, and double precision mode at a pixel rate up to 20 MHz.

ORDER CODES

Part Number	Temperature Range	Package
STV3200CP	0 to 70°C	DIP 40
STV3200CV	0 to 70°C	PQFP44

3208-01.TBL

PIN CONNECTIONS

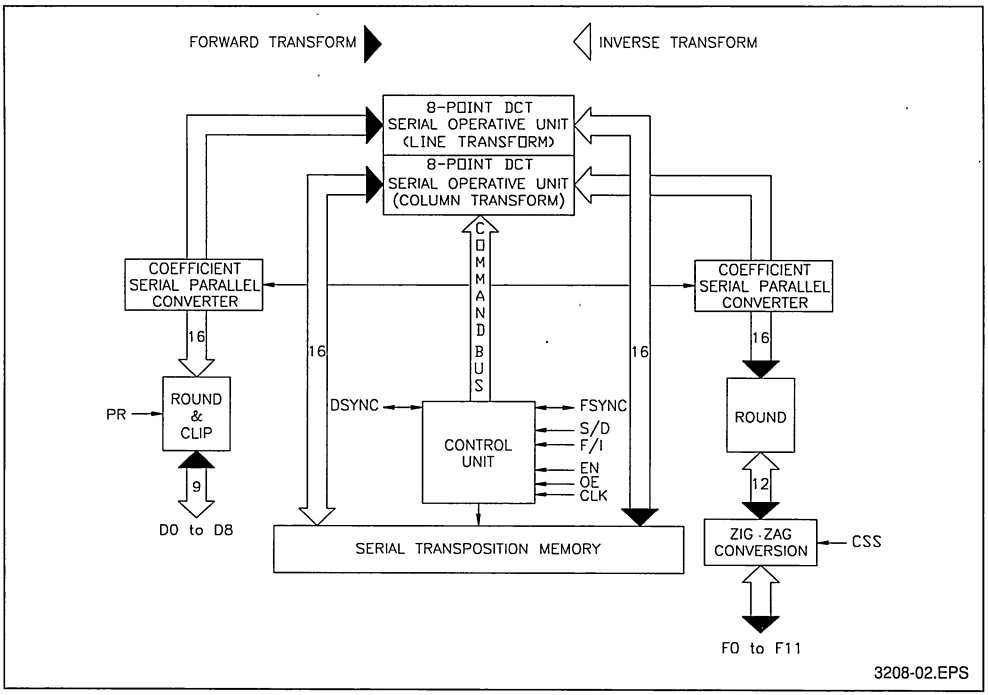


PIN IDENTIFICATION

Pin Number	Symbol	Type	Function / Description
4-12	D0 to D8	I/O	Pixel data bus
13	PR	Input	Pixel range selection
14	DSYNC	I/O	Pixel block synchronization signal
17	S/D	Input	Single/double precision selection
18	F/I	Input	FDCT/IDCT selection
19	OE	Input	Output three-state control
21	CLK	Input	Clock signal
22	EN	Input	Clock enable signal
23	TEST	Input	Test mode selection
26	CSS	Input	Zig Zag selection
27	FSYNC	I/O	Coefficient block synchronization signal
28-39	F0 to F11	I/O	Coefficient data bus
1,20,25	VSS	Power	Ground
15,40	VCC	Power	Power supply
2,3,24	NC		Not Connected

3208-02.TBL

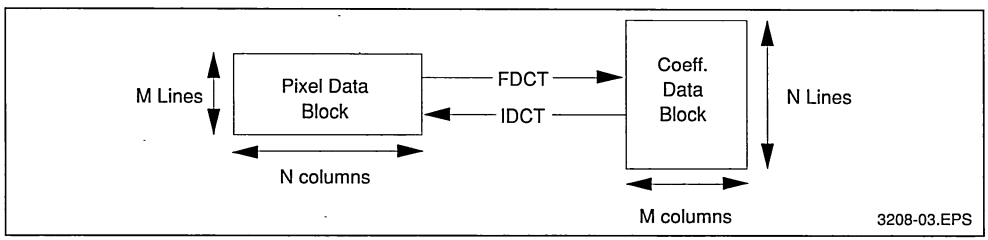
FUNCTIONAL BLOCK DIAGRAM



FUNCTIONAL DESCRIPTION

1. EQUATIONS

Figure 1



The STV3208 performs 8 x 8 two dimensional Discrete Cosine Transform according to the following formula:

Equations for 9-bit PIXEL DATA (PR pin set to low) :

FORWARD TRANSFORM EQUATION :

$$F(u,v) = \text{Round}\left[\frac{1}{4}C(u)\,C(v)\sum_{i=0}^{7}\sum_{j=0}^{7}D(i,j)\cos\frac{(2\cdot i+1)\,u\,\pi}{16}\cos\frac{(2\cdot j+1)\,v\,\pi}{16}\right]$$

INVERSE TRANSFORM EQUATION :

$$D(i,j) = \text{Round}\left[\frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)\ C(v)\ F(u,v)\ \cos\frac{(2\cdot i+1)\,u\pi}{16}\ \cos\frac{(2\cdot j+1)\,v\pi}{16}\right]$$

Where $C(u) = \frac{1}{\sqrt{2}}$ if $u = 0$

$= 1$ otherwise

Equations for 8-bit PIXEL DATA (PR pin set to high) :

FORWARD TRANSFORM EQUATION :

$$F(u,v) = \text{Round}\left[\frac{1}{2}C(u)\ C(v)\sum_{i=0}^{7}\sum_{j=0}^{7} D(i,j)\ \cos\frac{(2\cdot i+1)\,u\,\pi}{16}\ \cos\frac{(2\cdot j+1)\,v\,\pi}{16}\right]$$

INVERSE TRANSFORM EQUATION :

$$D(i,j) = \text{Round}\left[\frac{1}{8}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)\ C(v)\ F(u,v)\ \cos\frac{(2\cdot i+1)\,u\pi}{16}\ \cos\frac{(2\cdot j+1)\,v\pi}{16}\right]$$

Where $C(u) = \frac{1}{\sqrt{2}}$ if $u = 0$

$= 1$ otherwise

2. DATA FLOW ORDERING

The pixel block is scanned column by column (ORDER 1) or line by line (ORDER 2). If CSS is high, the coefficient block is scanned with a zig zag order. Figure 2 shows the relation between pixels order and coefficient order.

Figure 2a : Data Ordering (CSS high)

	PIXEL ORDER									COEFFICIENT ORDER							
	1	9	17	25	33	41	49	57		1	2	6	7	15	16	28	29
	2	10	18	26	34	42	50	58		3	5	8	14	17	27	30	43
	3	11	19	27	35	43	51	59		4	9	13	18	26	31	42	44
ORDER 1	4	12	20	28	36	44	52	60	⇔	10	12	19	25	32	41	45	54
	5	13	21	29	37	45	53	61		11	20	24	33	40	46	53	55
	6	14	22	30	38	46	54	62		21	23	34	39	47	52	56	61
	7	15	23	31	39	47	55	63		22	35	38	48	51	57	60	62
	8	16	24	32	40	48	56	64		36	37	49	50	58	59	63	64
	1	2	3	4	5	6	7	8		1	3	4	10	11	21	22	36
	9	10	11	12	13	14	15	16		2	5	9	12	20	23	35	37
	17	18	19	20	21	22	23	24		6	8	13	19	24	34	38	49
ORDER 2	25	26	27	28	29	30	31	32	⇔	7	14	18	25	33	39	48	50
	33	34	35	36	37	38	39	40		15	17	26	32	40	47	51	58
	41	42	43	44	45	46	47	48		16	27	31	41	46	52	57	59
	49	50	51	52	53	54	55	56		28	30	42	45	53	56	60	63
	57	58	59	60	61	62	63	64		29	43	44	54	55	61	62	64

3208-03.TBL

If CSS is low, the coefficient block is scanned line by line and the pixel block is scanned column by column, or the coefficient block is scanned column by column and the pixel block is scanned line by line.

Figure 2b : Data Ordering (CSS low)

	PIXEL ORDER									COEFFICIENT ORDER								
	1	9	17	25	33	41	49	57		1	2	3	4	5	6	7	8
	2	10	18	26	34	42	50	58		9	10	11	12	13	14	15	16
	3	11	19	27	35	43	51	59		17	18	19	20	21	22	23	24
ORDER 1	4	12	20	28	36	44	52	60	⇔	25	26	27	28	29	30	31	32
	5	13	21	29	37	45	53	61		33	34	35	36	37	38	39	40
	6	14	22	30	38	46	54	62		41	42	43	44	45	46	47	48
	7	15	23	31	39	47	55	63		49	50	51	52	53	54	55	56
	8	16	24	32	40	48	56	64		57	58	59	60	61	62	63	64
	1	2	3	4	5	6	7	8		1	9	17	25	33	41	49	57
	9	10	11	12	13	14	15	16		2	10	18	26	34	42	50	58
	17	18	19	20	21	22	23	24		3	11	19	27	35	43	51	59
ORDER 2	25	26	27	28	29	30	31	32	⇔	4	12	20	28	36	44	52	60
	33	34	35	36	37	38	39	40		5	13	21	29	37	45	53	61
	41	42	43	44	45	46	47	48		6	14	22	30	38	46	54	62
	49	50	51	52	53	54	55	56		7	15	23	31	39	47	55	63
	57	58	59	60	61	62	63	64		8	16	24	32	40	48	56	64

3208-04.TBL

3. DATA FORMAT

Coefficients format is 12-bit 2's complement, corresponding to the range -2048 to 2047.

There are 2 possible ranges for pixel data :

9-bit two's complement magnitude
(see Figure 3)

The pixel data range is -256 to +255. In this case the PR pin must be set to 0 for IDCT. D8 is the most significant bit and D0 the least significant bit for the pixel data. A clipping to the range -256 to +255 is performed before outputting reconstructed pixels after an IDCT.

8-bit two's complement magnitude
(see Figure 4)

Pixel data range is -128 to +127. In this case D0 must be set to 0 and the PR Pin must be set to 1 for IDCT. D8 is the most significant bit and D1 the least significant bit for the pixel data. A clipping to the range -128 to +127 is performed before outputting reconstructed pixels after an IDCT.

This mode may be used for intra picture coding. In this case, pixel data range is 0 to 255. For a FDCT, the most significant bit of input pixel data (D8) must be inverted before entering the chip. This is equivalent to substract 128 to the input pixel data. Note that this operation will only have effect on the DC value F(0,0). For an IDCT, the most significant bit of output pixel data (D8) must be inverted. This is equivalent to add 128 to the output pixel data.

Figure 3

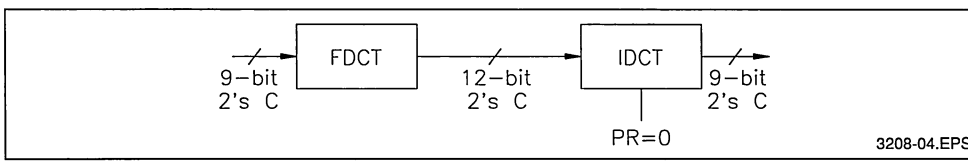
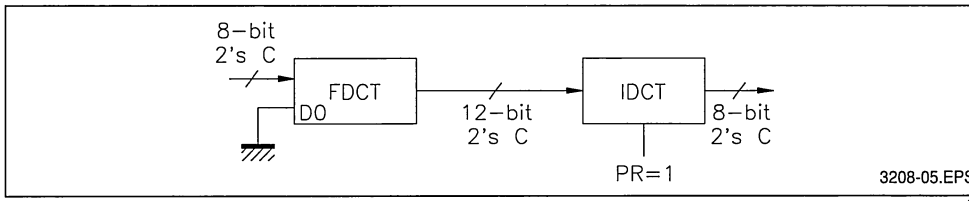


Figure 4



4. BLOCK FLOW

Depending on the application, blocks may be entered in different way.

Latent period :

The latent period between input data and corresponding output results is 167 clock cycles (if FDCT is selected) or 163 clock cycles (if IDCT is selected) in single precision mode (S/D pin set to 1). This means that the first data of the resulting block is provided 137 clock cycles (if FDCT is selected) or 135 clock cycles (if IDCT is selected) in double precision mode (S/D pin set

to 0).

Latency	Forward DCT	Inverse DCT
S/D = 1 Single Precision	167 Cycles	163 Cycles
S/D = 0 Double Precision	137 Cycles	135 Cycles

Synchronization signals :

An input block synchronization signal must be provided. The input pin for this signal is DSYNC if FDCT is selected and FSYNC if IDCT is selected.This signal is active low and must not be active more than one clock cycle and during the first clock cycle after power-up.This signal must be active with the first data of each input block or group of blocks.

An output block synchronization signal is provided. The output pin for this signal is FSYNC if FDCT is selected and DSYNC if IDCT is selected. This signal is active with the first data of each output block or group of blocks.

The output synchronization signal is equal to the input synchronization signal delayed from the latent period (see Figure 5).

CONTINUOUS BLOCK FLOW

Inputs data are fed continously with one new item data at each clock cycle and output data is provided continously with one new result data item at each clock cycle.

The input synchronization signal can be provided for each input block. In this case the output synchronization pulse is provided for each output block (Figure 6). An other way is to provide a synchronization pulse only for the first block of a group of blocks. In this case, only one synchronization pulse is provided for the first output block (Figure 7).

Figure 5

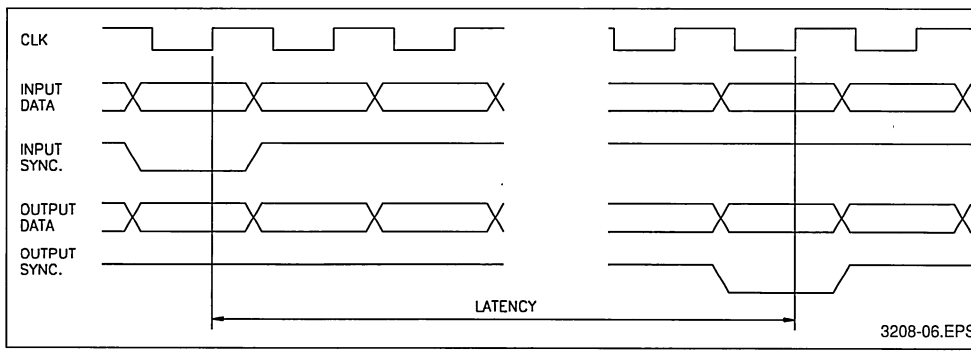


Figure 6 : Continuous Block Flow 1

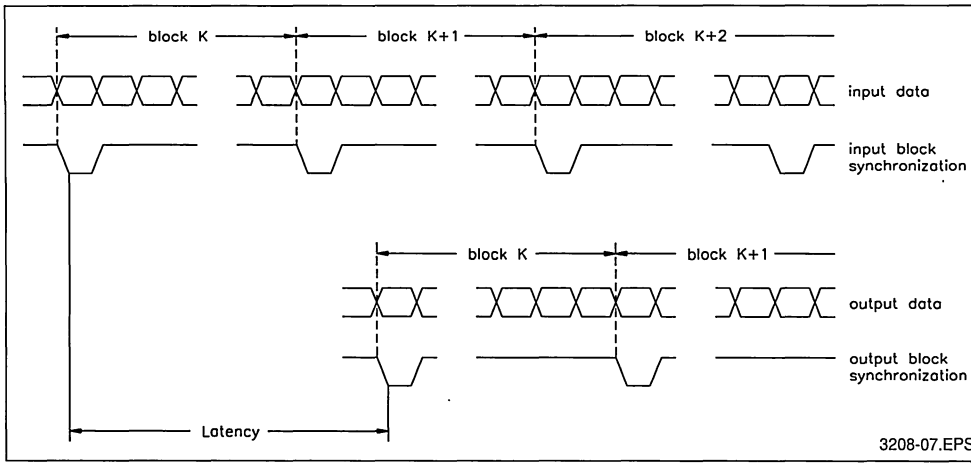
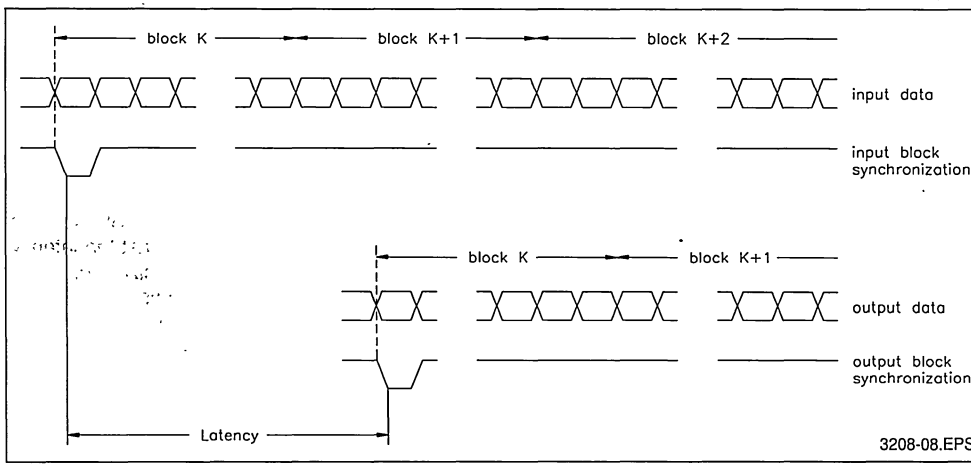


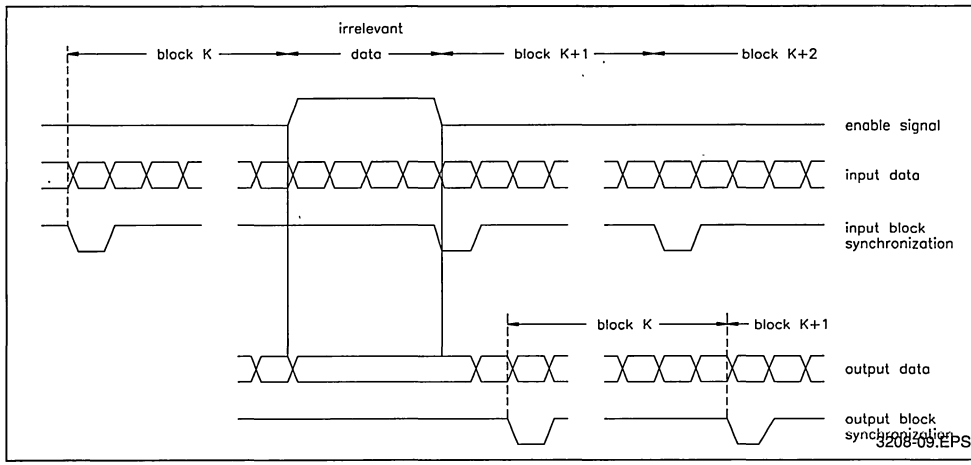
Figure 7 : Continuous Block Flow 2



CONTINUOUS BLOCK FLOW WITH BYPASS OF IRREVELANT DATA

It is possible to process a block flow including irrelevant data (corresponding to line suppression for example) as if it was a continous block flow. One way is to stop the clock signal during the irrelevant data occurence. Another way is to use the Clock Enable Signal (EN) which allows to stop the chip internal clock during irrelevant data occurrence (see Figure 8)

Figure 8 : Continuous Block Flow with Irrelevant Data



BURST BLOCK FLOW (see Figure 9)

Single blocks (or groups of block) may not be continous. In other words, delay cycles between two blocks (or groups of block) may exist. During these delay cycles, the clock is still running and the chip continues to perform computations. The constraint is that the internal pipe line must not be broken when the new block occurs.To take this constraint into account, the number of delay cycles (NC) must respect one of the following conditions :

1 - the number of delay cycles (NC) is greater than or equal to the latency. In this case the pipe line is empty (all the relevant data has been output) when a new input block processing starts.

2 - the number of delay cycles (NC) is a multiple of 64. In this case, the input data always remains synchronous with the internal pipe line.

MIXED FDCT/IDCT (see Figure 10)

In some low frequency application, it could be cost effective to use only one chip to compute all the DCT required by the coding scheme. Blocks must be fed in a burst fashion with at least the latency time between the last pixel of input pixels for FDCT and the first pixel of input coefficients for IDCT. The same delay must be respected between the last pixel of input coefficients for IDCT and the first pixel of input pixels for FDCT.

Figure 9 : Burst Block Flow

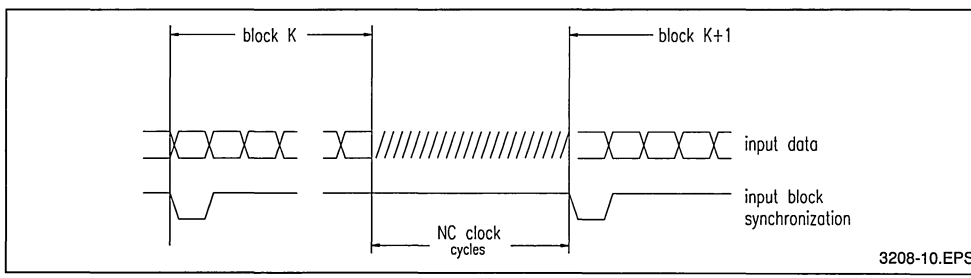
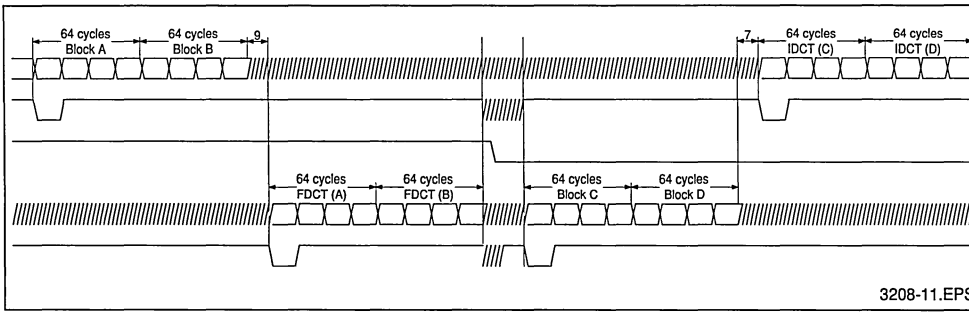


Figure 10 : Mixed 8 x 8 FDCT/IDCT Example Waveforms (double precision)



PRECISION SELECTION

For single precision mode, the S/D pin must be high. In this case, the maximum rating for pixel is 27 MHz. For double precision mode, the S/D pin must be low. In this case, the maximum rating for pixel is 20 MHz.

5. PINS DESCRIPTION

CLK : Clock signal

DATA PINS

D0 to D8 : 9-bit bidirectional pixel data bus pins. Direction is programmed by the F/I pin :

F/I state	D0 to D8 Direction
High	Input
Low	Output

Data is loaded (when input) or settled (when output) on rising edge of CLK. D0 is the least significant bit and D8 the most significant one. Note that for the optimized mode for 8-bit 2's C pixel data, D1 is the least significant bit and D0 must be set to 0.

MSB								LSB	
D8	D7	D6	D5	D4	D3	D2	D1	D0	Pin
-256	128	64	32	16	8	4	2	1	Weight

DSYNC : pixel data block synchronization signal. This pin is bidirectional with the direction programmed by the F/I pin (like D0 to D8). DSYNC is active (low level during one clock cycle only) with the first pixel data of a block (or a group of blocks).

F0 to F11 : 12-bit bidirectional coefficient data bus pins. Direction is programmed by the F/I pin :

F/I state	F0 to F11 direction
High	Output
Low	Input

Data is loaded (when input) or settled (when output) on rising edge of CLK. F0 is the least significant bit and F11 the most significant one.

MSB											LSB	
F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0	Pin
-2048	1024	512	256	128	64	32	16	8	4	2	1	Weight

FSYNC : coefficient data block synchronization signal. This pin is bidirectional with the direction programmed by the F/I pin (like F0 to F11). FSYNC is active (low level during one clock cycle only) with the first coefficient data of a block (or a group of blocks).

CONTROL PINS

F/I : forward or inverse selection. When F/I is high, forward DCT is performed, when F/I is low, inverse DCT is performed.

S/D : single or double precision. When S/D is high, single precision is selected. When S/D is low, double precision is selected.

CSS : coefficient scanning selection. When CSS is high, zig zag scanning of coefficient block is selected. When CSS is low, row scanning of coefficient block is selected.

PR : pixel range selection. If PR is low, pixel range is -256 to +255. If PR is high, pixel range is -128 to +127.

OE : output enable. This signal is active low. When OE is high, all outputs (defined by the F/I pin state) are forced to the high impedance state.

EN : enable. This signal is active low. When EN is high, internal states of the chip are frozen. When EN becomes low, execution restarts. EN must go to high state when CLK is high.

State	Function
F/I is High F/I is Low	Forward DCT Inverse DCT
S/D is High S/D is Low	Single Precision Double Precision
CSS is High CSS is Low	Zig Zag Scanning of coefficients Row Scanning of coefficients
PR is High PR is Low	8-bit 2's C Pixel Data 9-bit 2's C Pixel Data
OE is High OE is Low	High Impedance Outputs Outputs Active
EN is High EN is Low	Internal Clock is stopped Internal Clock runs

POWER SUPPLY AND GROUND PINS

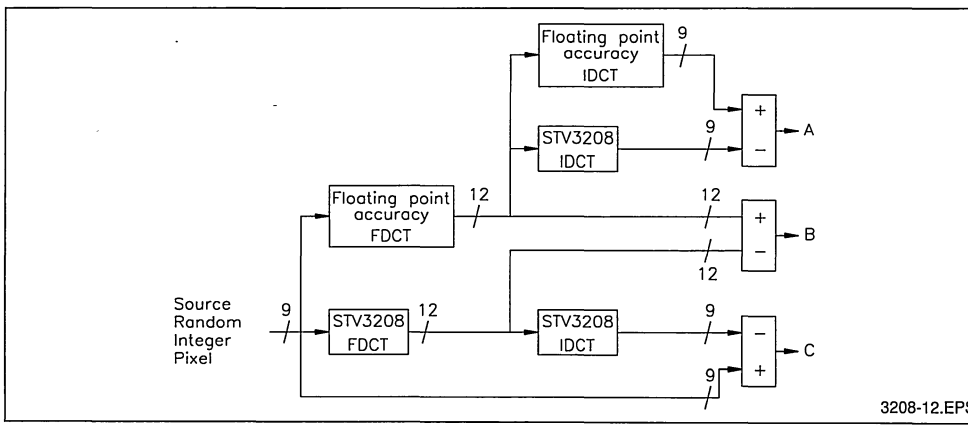
V_{CC} : +5 Volt power supply

V_{SS} : ground potential

OTHERS : Test. This pin is reserved and must be low in normal mode.

6. ACCURACY CHARACTERISTICS

The accuracy characteristics have been measured according to the following scheme :



A :

Characteristics of IDCT. Error between the IDCT computed with 64-bit floating point accuracy and the IDCT computed by the STV3208 is measured. Measures have been done according to the CCITT WGXV method

	Single Precision	Double Precision
Peak Error	1	1
Peak Mean Square Error	0.0403	0.0258
Overall Mean Square Error	0.0287	0.0200
Peak Mean Error	0.0125	0.0041
Overall Mean Error	0.0050	0.0000062

	9-bit pixels		8-bit pixels	
	Single Precision	Double Precision	Single Precision	Double Precision
Exact value	97.1 %	98.0 %	99.96 %	99.97 %
Errors of ± 1 LSB	2.9 %	2.0 %	0.04 %	0.03 %
Errors of ± 2 LSB	0 %	0 %	0 %	0 %

B :

Characteristics of FDCT. Error between the FDCT computed with 64-bit floating point accuracy and the FDCT computed by the STV3208 is measured.

	9-bit pixels		8-bit pixels	
	Single Precision	Double Precision	Single Precision	Double Precision
Exact value	93.6 %	96.9 %	93.6 %	96.9 %
Errors of ± 1 LSB	6.4 %	4.1 %	6.4 %	4.1 %
Errors of ± 2 LSB	0 %	0 %	0 %	0 %

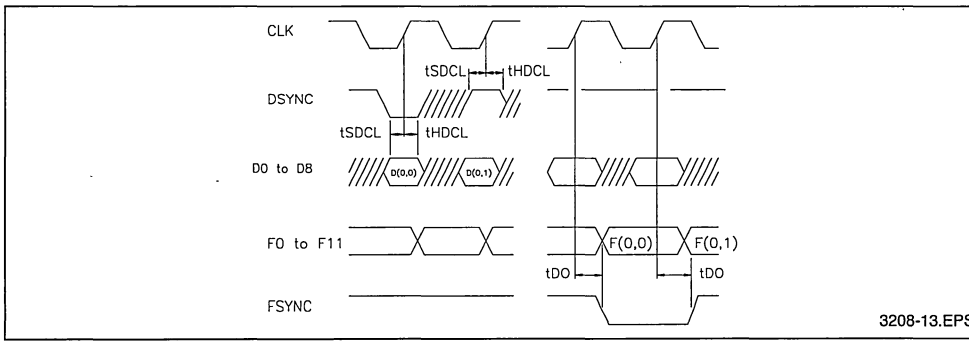
C :

Characteristics of FDCT followed by an IDCT. Error between the source picture and the FDCT computed by the STV3208 followed by an IDCT computed by the STV3208 is measured.

	9-bit pixels		8-bit pixels	
	Single Precision	Double Precision	Single Precision	Double Precision
Exact value	89.3 %	90.6 %	99.88 %	99.92 %
Errors of ± 1 LSB	10.7 %	9.4 %	0.12 %	0.08 %
Errors of ± 2 LSB	0 %	0 %	0 %	0 %

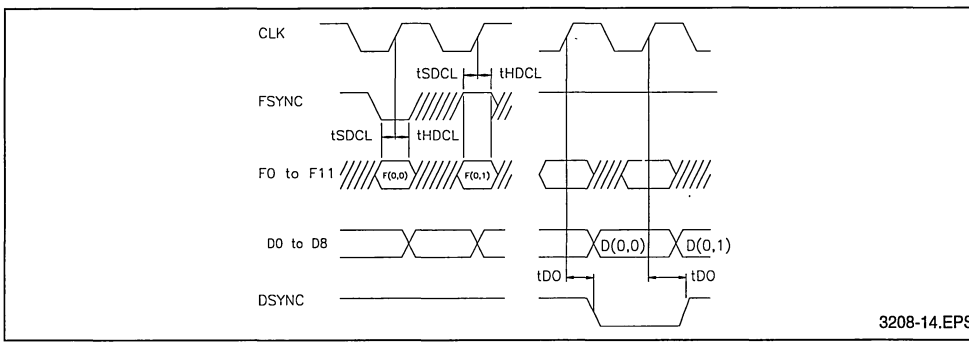
TIMING WAVEFORMS

Synchonization Signals Timing Diagram for a Forward Transform



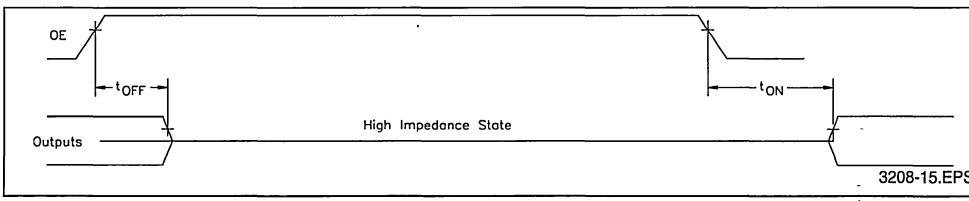
Note : FSYNC will be in unknown state after the power up during a count of cycles equal to the latency.

Synchronization Signals Timing Diagram for an Inverse Transform

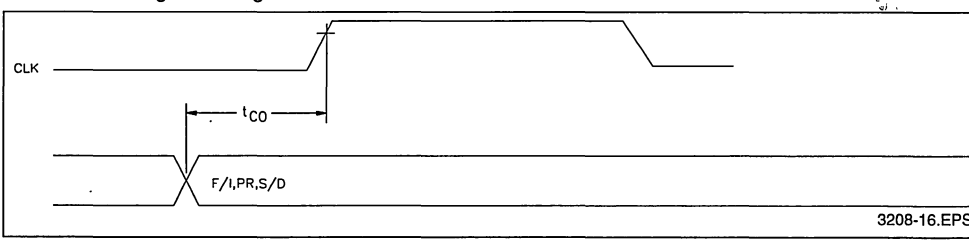


Note : DSYNC will be in unknown state after the power up during a count of cycles equal to the latency.

Output Enable Signal Timing Waveforms

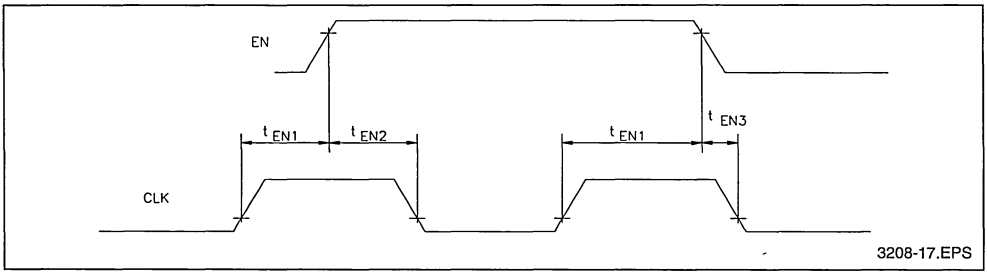


Control Static Signal Timing Waveforms



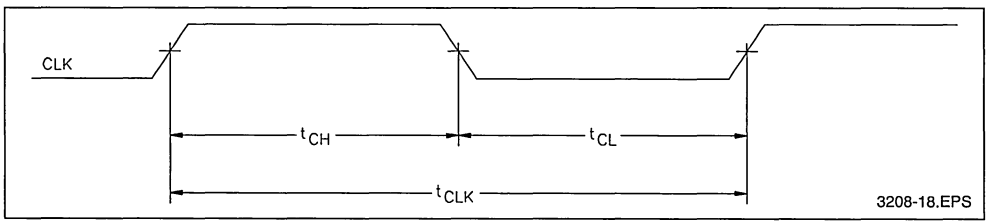
TIMING WAVEFORMS (continued)

Enable Signal Timing Waveforms

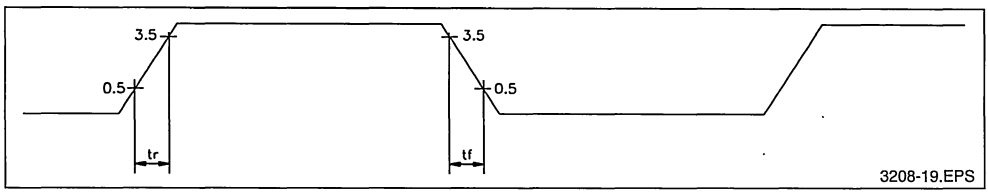


Note : EN signal must change from low to high level during the high level of CLK signal.

Clock Timing Waveforms



Output Timing Waveforms



ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Supply voltage (V_{CC}) : 6 Volts

Operating temperature range : 0 to 70 °C

DC ELECTRICAL CHARACTERISTICS

Operating Conditions : V_{SS} = 0 Volt, T_A = 0 to 70 °C, V_{CC} = 5 V ± 5% unless otherwise specified

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{CC}	Operating Voltage		4.75		5.25	V
I_{CC}	Supply current : F_{CLK} = 27 MHz F_{CLK} = 0 MHz	C_{LOAD} = 50 pF on all outputs. All inputs at V_{CC} or V_{SS}			 150 1	 mA mA
V_{IL} V_{IH}	Input Voltage Level (except CLK) Logic Low Logic High	V_{CC} = 5 ± 0.25 V	 2		0.8	 V V
$V_{IL(CLK)}$ $V_{IH(CLK)}$	Clock Signal Logic Low Logic High		 2.5		0.5	 V V
	High Impedance input leakage : I/O Buffers Input Buffers	V_{IN} = V_{SS} to V_{CC}	 -5 -1		 +5 +1	 µA µA
V_{OL} V_{OH}	Output Voltage Level : Logic Low I_{LOAD} = 500µA Logic High I_{LOAD} = -500µA	V_{CC} = 4.75 V	 2.7		0.4	 V V
C_{IN}	Input capacitance	V_{OFFSET} = 2.5 V, f = 1 MHz			10	pF

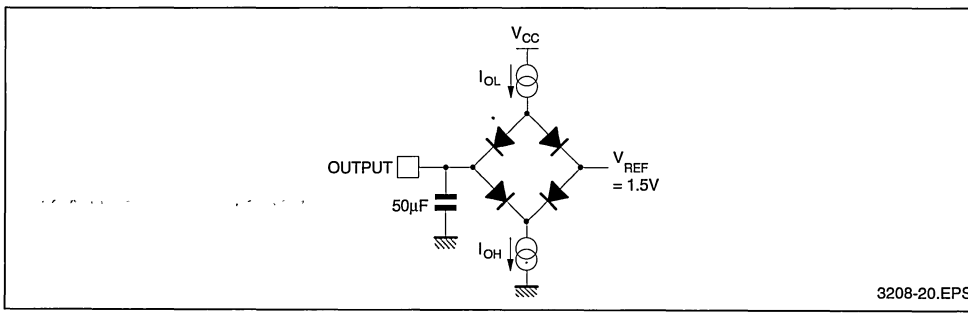
3208-05.TBL

AC ELECTRICAL CHARACTERISTICS

Operating conditions : V_{SS} = 0 Volt, T_A = 0 to 70 °C, V_{CC} = 5 V ± 5 % unless otherwise specified

Outputs Loads : Capacitance = 50 pF, Current Logic Low = 500µA

Test Load on Outputs :



3208-20.EPS

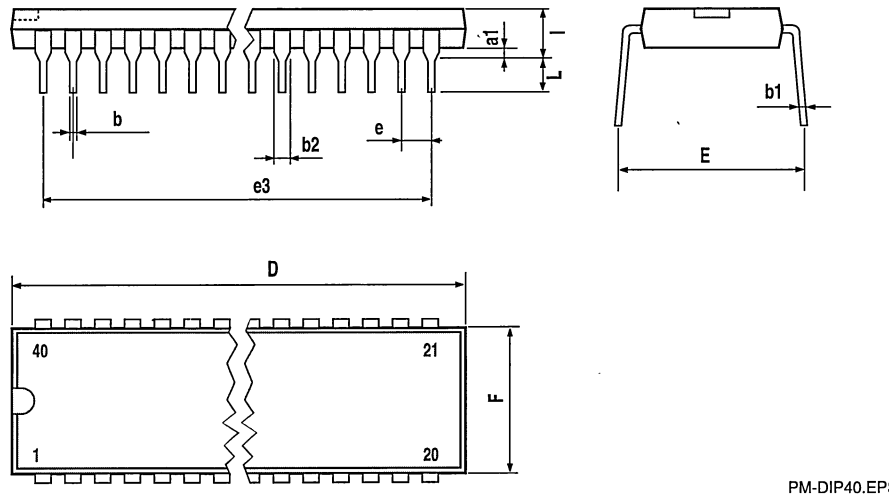
Timings are measured between threshold voltage of 1.5 V unless otherwise specified.

Symbol	Parameter	Min.	Typ.	Max.	Unit
t_R	Rising time from 0.5 to 3.5 V			10	ns
t_F	Falling time from 3.5 to 0.5 V			10	ns
t_{CH}	Clock High Pulse Width S/D = 1 S/D = 0	 15 24			 ns ns
t_{CL}	Clock Low Pulse Width S/D = 1 S/D = 0	 15 24			 ns ns
t_{CLK}	Clock Cycle Duration S/D = 1 S/D = 0	 37 50			 ns ns
t_{SDCL}	Data Setup Time from CLK ↑	8			ns
t_{HDCL}	Data Hold Time from CLK ↑	0			ns
t_{DO}	Output Data Delay from CLK ↑			15	ns
t_{EN1}	Enable Hold Time from CLK ↑	0			ns
t_{EN2}	Enable Rising Edge Setup Time from CLK ↓	5			ns
t_{EN3}	Enable Falling Edge Setup Time from CLK ↓ S/D = 1 S/D = 0	 0 0			 ns ns
t_{OFF}	Delay from OE ↑ to Output going to High Impedance State			15	ns
t_{ON}	Delay from OE ↓ to Output going to High or Low State			15	ns
t_{CO}	F/I , CSS, PR, S/D Setup Time from Beginning of Input Stream	100			ns

3208-06.TBL

PACKAGE MECHANICAL DATA

40 PINS - PLASTIC DIP

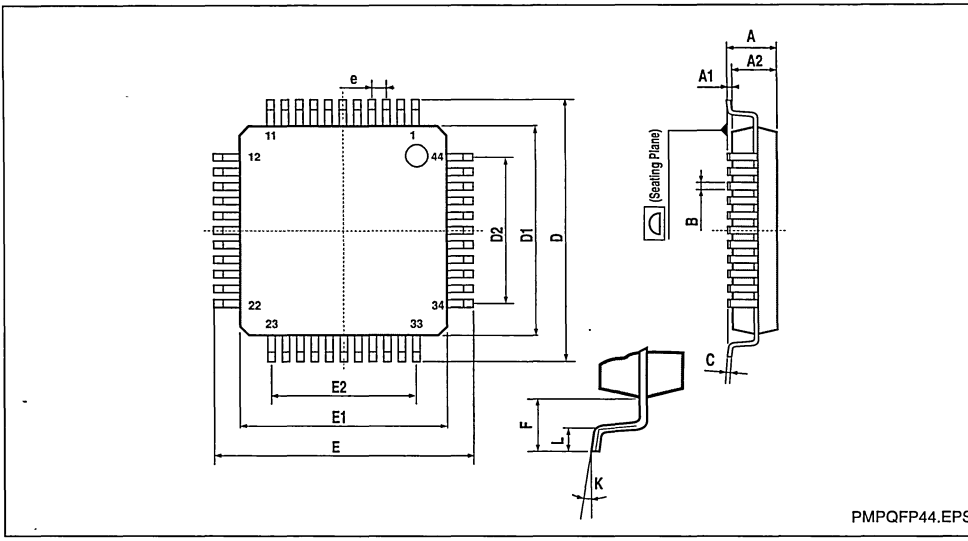


Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
a1		0.63			0.025	
b		0.45			0.018	
b1	0.23		0.31	0.009		0.012
b2		1.27			0.050	
D			52.58			2.070
E	15.2		16.68	0.598		0.657
e		2.54			0.100	
e3		48.26			1.900	
F			14.1			0.555
i		4.445			0.175	
L		3.3			0.130	

DIP40.TBL

PACKAGE MECHANICAL DATA

44 PINS - PLASTIC QUAD FLAT PACK



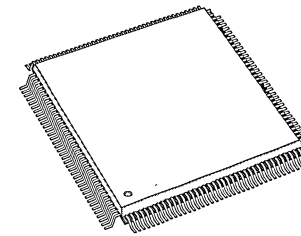
Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A			3.40			0.134
A1	0.25			0.01		
A2	2.55	2.80	3.05	0.10	0.11	0.12
B	0.35		0.50	0.014		0.020
C	0.13		0.23	0.005		0.009
D	16.95	17.20	17.45	0.667	0.677	0.687
D1	13.90	14.00	14.10	0.547	0.551	0.555
D2		10.00			0.394	
e		1.00			0.039	
E	16.95	17.20	17.45	0.667	0.677	0.687
E1	13.90	14.00	14.10	0.547	0.551	0.555
E2		10.00			0.394	
F		1.60			0.063	
K	0^o (min.), 7^o (max.)					
L	0.65	0.80	0.95	0.025	0.031	0.037

PQFP44.TBL

MOTION ESTIMATION PROCESSOR

ADVANCE DATA

- PIXEL RATE FROM 0 UP TO 18MHz
- BLOCK SIZE : 8 x 4n, 16 x 4n
- MAXIMUM DISPLACEMENT +7/-8 PIXELS HORIZONTAL AND VERTICAL
- COMPUTATION OF THE MOTION VECTOR AND MINIMUM DISTORTION
- RANDOM ACCESS TO THE 256 DISTORTIONS
- 8-BIT UNSIGNED INPUT PIXEL
- 4-BIT 2'S COMPLEMENT HORIZONTAL DISPLACEMENT
- 4-BIT 2'S COMPLEMENT VERTICAL DISPLACEMENT
- 16-BIT UNSIGNED DISTORTION VALUES
- CLOCK FREQUENCY = INPUT RATE
- FULLY TTL AND CMOS COMPATIBLE
- CMOS TECHNOLOGY
- SINGLE + 5 VOLT POWER SUPPLY
- MAXIMUM POWER DISSIPATION : 2 WATTS AT 18MHz CLOCK RATE



PQFP144
(Plastic Package)

ORDER CODE : STI3220CV

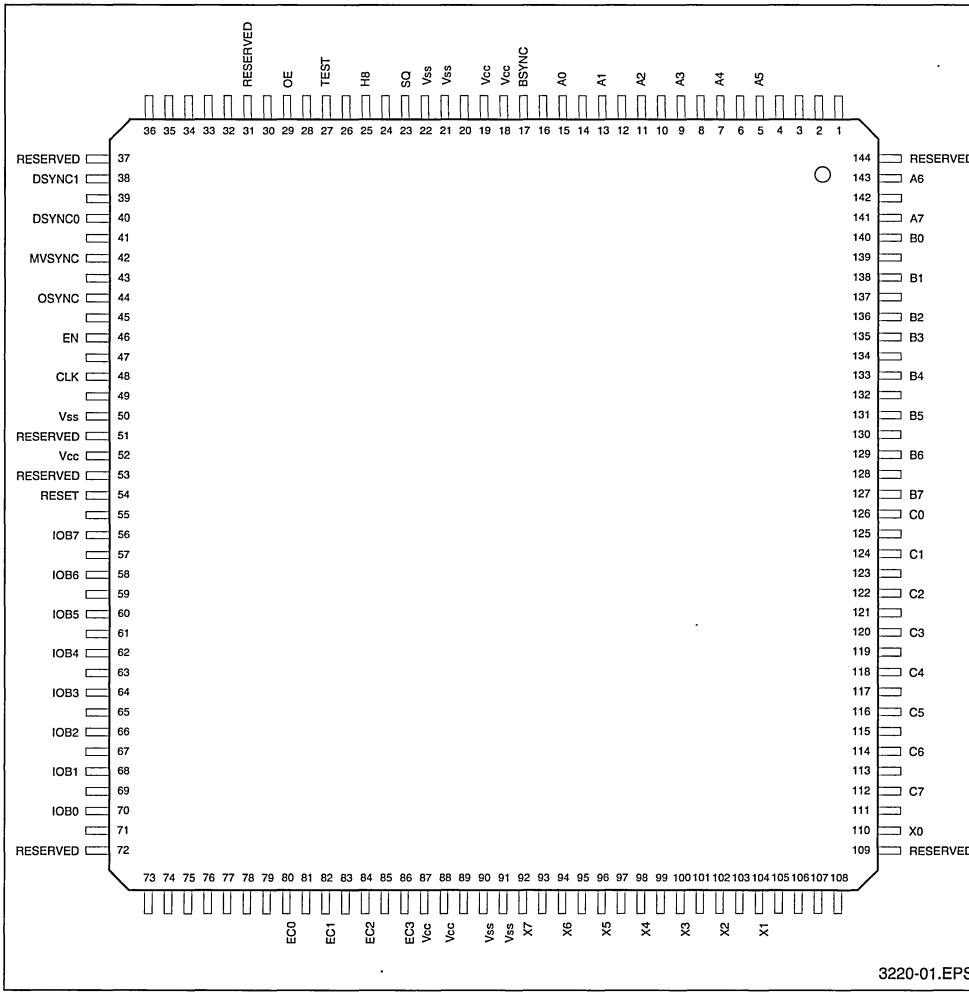
DESCRIPTION

The Real Time Motion Estimation Chip is a dedicated circuit for motion estimation at video rate.
The chip is optimized to compute the displacement vector of 8 x 4n or 16 x 4n blocks in a search window (SW) defined by a maximum horizontal and vertical displacement of +7/-8 pixels corresponding to 256 different vectors.
The chip computes 256 distortions for each block according to the MAE criterion.
The minimum distortion and the corresponding vector are calculated on chip. A random access to all distortions allows to implement more elaborate algorithms at the system level.
Displacement vectors of +15/-16 pixels are accommodated with a single chip for lower rates (time multiplexed) or with several chips for higher rates (spatial multiplexed).
The following table shows different application examples for the motion estimation chip :

Picture format	Pixel Rate	Block Size	Maximum Displacement	Minimum Chip Clock Frequency	Number of chips
CIF 10Hz	1.01MHz	8 x 4n, 16 x 4n	+7 / -8	1.01MHz	1
			+15 /-16	4.07MHz	1
CIF 30Hz	3.04MHz	8 x 4n, 16 x 4n	+7 / -8	3.04MHz	1
			+15 /-16	12.2MHz	1
TV 25Hz	13.5MHz	8 x 4n, 16 x 4n	+7 / -8	13.5MHz	1
			+15 /-16	13.5MHz	4

3220-01.TBL

PIN CONNECTIONS



3220-01.EPS

1. PIN LIST

Pin Number	DESCRIPTION
5	A5
7	A4
9	A3
11	A2
13	A1
15	A0
17	BSYNC
18, 19	V_{CC}
21, 22	V_{SS}
23	SQ
25	H8
27	TEST
29	OE
31	Reserved
37	Reserved
38	DSYNC1
40	DSYNC0
42	MVSYNC
44	OSYNC
46	EN
48	CLK
50	V_{SS}
51	Reserved
52	V_{CC}
53	Reserved
54	RESET
56	IOB7
58	IOB6
60	IOB5
62	IOB4
64	IOB3
66	IOB2
68	IOB1
70	IOB0
72	Reserved

Pin Number	DESCRIPTION
80	EC0
82	EC1
84	EC2
86	EC3
87, 88	V_{CC}
90, 91	V_{SS}
92	X7
94	X6
96	X5
98	X4
100	X3
102	X2
104	X1
109	Reserved
110	X0
112	C7
114	C6
116	C5
118	C4
120	C3
122	C2
124	C1
126	C0
127	B7
129	B6
131	B5
133	B4
135	B3
136	B2
138	B1
140	B0
141	A7
143	A6
144	Reserved

3220-02.TBL

All other pins are not connected

2. INTRODUCTION

The STI3220 circuit is intended for use in video compression systems where motion estimation is employed. For example, CCITH.261 and ISO(MPEG) compatible video compression systems need to perform motion estimation. Potential applications are numerous and include, for example, systems concerned with image transmission and/or storage, e.g videophone, videoconference, digital VTR, multimedia computer, etc.

2.1 Motion Estimation Basics

Motion estimation is one of the techniques that play a role in video picture compression. The basic idea arises from a common sense observation : in a video sequence, successive frames are likely to represent the same details, with little difference between one frame and the next. A sequence showing moving objects over a still background is a good example. Important data compression can be effected if each component of a frame is represented by its difference with the most similar component - the **predictor** - in the previous frame, and by a vector - the **motion vector** - expressing the relative position of the two components. If an actual motion exists between the two frames, the difference may be null or very small. The original component can be reconstructed from the difference, the motion vector, and the previous frame. Motion estimation is the process of finding a good (if not the best) template for prediction and the corresponding vector. It does not in itself compress data but provides the basic information enabling data compression to be effected.

Several motion estimation techniques exist but the most popular one is based on a block by block processing and for this reason is called **block matching.**

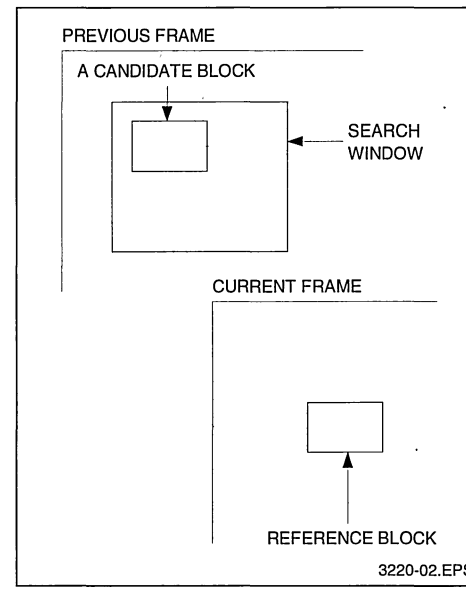
For block matching processing, the frame to be compressed is positioned into blocks which are processed individually. For a given block - the **reference block** - a predictor is searched for among candidate blocks of equal size included in a surrounding rectangular region in the previous frame - the **search window** (see Figure 1). The reference block is compared with the candidate blocks. The result of a comparison is called a **distortion** and serves as a measure of the similarity of the two blocks. The candidate block corresponding to the minimum distortion is the best match, and this block along with the vector relating it to the reference block, form the predictor.

When all the blocks included in the search area are compared with the reference block, the process is called **full search block matching.**

To compare blocks, several criterion exist, among which the **Mean Absolute Error (MAE),** is the most often used because it offers a good trade-off between complexity and efficiency.

The STI3220 performs full search block matching with the MAE criterion at pixel rates up to 18 MHz.

Figure 1 : Block matching notation



2.2 Notation

In Figure 2, the notation used throughout this document is illustrated.

The reference block, referred to as X, is M rows high and N columns wide. A pixel of block X situated at the intersection of row i and column j is denoted Xi,j. For block X, numbering of rows and columns starts at 0.

For the STI3220, M=8 or 16 and N is a multiple of 4 and is denoted by N=4n.

The search window, referred to as Y, is 15 pixels larger than X in each direction, leading to 256 candidate blocks, as many distortions and as many motion vectors. A pixel of search window Y situated at the intersection of row i and column j is denoted by Yi,j. For search window Y, numbering of rows and columns starts at -8.

A motion vector is expressed with two components, a vertical one and a horizontal one. The range of both components is [-8 +7]. Upward and left-hand components are negative; downward and right-hand components are positive.

The set of 256 distortions can be regarded as an array where the indices of a given distortion are precisely the components of the motion vector related to it. Distortion Di,j is related to the motion vector whose components are (i,j) and is equal to :

$$D_{i,j} = \sum_{m=1}^{M} \sum_{n=1}^{N} | X_{m,n} - Y_{m+i,n+j} |$$

3. FUNCTIONALITY

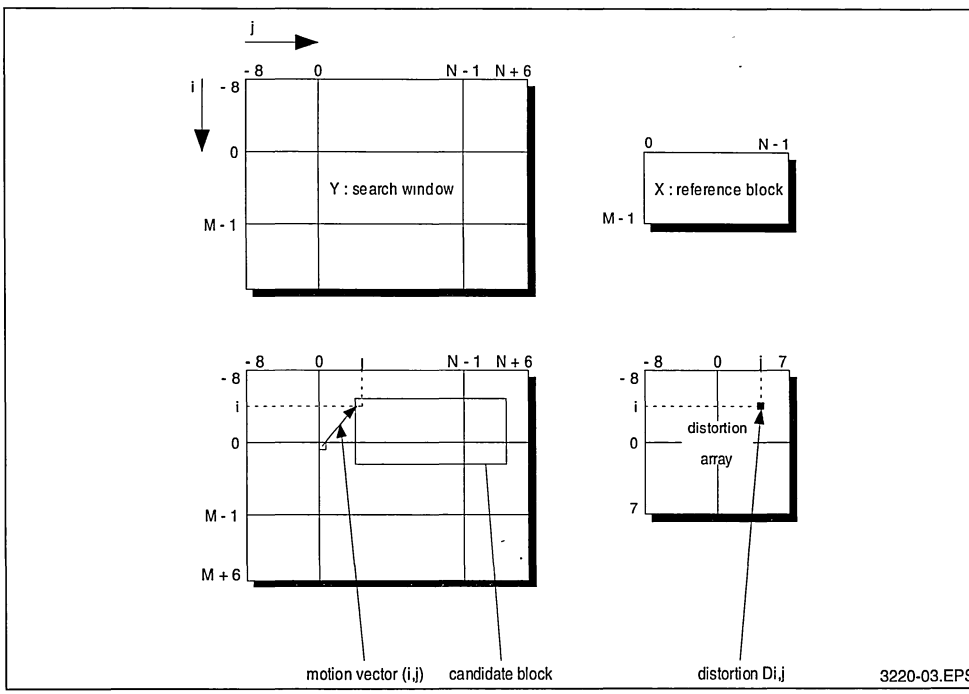
3.1 Overview

The STI3220 contains four 8-bit pixel ports, one to enter the reference block, and the other three to enter the search window. At each clock cycle, one pixel of the reference block and three (in the case of 8xN blocks) or two (in the case of 16xN blocks) pixels of the search window are entered in the circuit. For more details on how to enter pixels for the various block sizes, see section 4.

The STI3220 performs full search block matching with the MAE criterion over a search area expanding 15 pixels around the reference block, 8 pixels in the north and west directions and 7 pixels in the south and east directions. The reason for this asymetry is that the number of candidates is 256 so that the full range of the 8 bit motion vector can be used. The STI3220 can be used to perform motion estimation over larger search windows by using several circuits in parallel or one circuit in a time multiplex fashion. For more details concerning larger search windows, see section 5.

The STI3220 accomodates various block sizes. Block height is 8 or 16 and block width is a multiple of four. As internal accumulators are 16 bit wide, blocks of more than 256 pixels may cause overflow and generate erroneous results. Thus, the practical maximum sizes of blocks are 8x32 and 16x16. However, wider blocks can still be processed, using 7-bit pixels for example. For more details on how to set the block size, see section 4.9.

Figure 2 : Notation



The STI3220 provides control to manage the various cases when a block is situated near the edge of a picture. In such a situation, the search window is smaller and the motion vector set is reduced. There are 9 different cases depending on which edge/corner of the picture is concerned. For more details on how to set edge control, see section 4.8. For each reference block entered on the circuit, a motion vector and the corresponding minimum distortion are systematically delivered a few cycles after the last pixel of the reference block has been entered in the circuit. When several distortions are equal to the minimum, the circuit outputs the vector which has the highest priority according to a priority table given in section 4.6.

Results are delivered on an 8-bit bidirectional bus. The directionality of this bus is entirely controlled by the system. Motion vectors and minimum distortions are delivered through this bus. Also, the circuit can read in an address and write out the corresponding distortion. For more details on the management of this bus, see sections 4.6 and 4.7.

The STI3220 can be operated in an efficient pipeline mode where successive reference blocks are processed without any dead cycle between the last pixel of a block and the first pixel of the following block. In this mode, the circuit performs motion estimation at the video pixel rate, up to 18 MHz. For more details on modes of operation, see section 4.

3.2 Block Diagram

Figure 3 features the block diagram of the circuit.

The STI3220 is composed of a 256 processor systolic array which computes the distortions, a 256x16 SRAM where distortions are temporarily stored, a comparator which computes the minimum among the distortions stored in RAM, a bidirectional bus which is used to input addresses and to output motion vectors and the contents of the RAM, and a set of formatters which reorganize the data coming from the 4 8-bit pixel ports before feeding it to the processors.

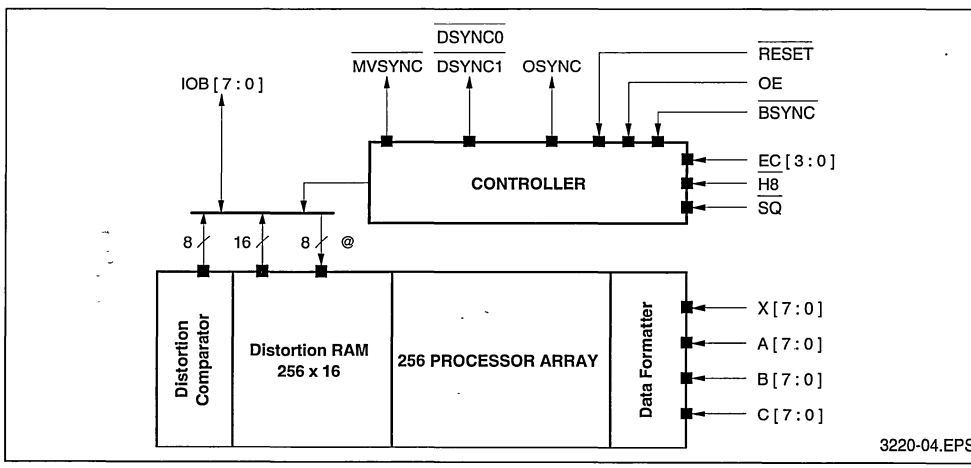
The 256 processors are all identical and work concurrently. Each one is dedicated to one distortion of the distortion array shown above. As soon as a set of computations is finished, the results are simultaneously transferred to the RAM so that a new set of computations can start without any dead cycle.

The RAM can be read from the system at any time, except when a minimum computation is in process. The address of a distortion is precisely the vector related to it.

The comparator is composed of sixteen comparators computing in parallel, so that the 255 comparisons necessary to estimate the minimum distortion are carried out rapidly, in order to set the RAM free as soon as possible for external access. The comparator also stores the address of the minimum which is precisely the motion vector.

The directionality of the I/O bus is entirely controlled by the system through the OE (output enable) pin, so that the system can choose the moment when results are made available.

Figure 3 : STI3220 Block Diagram



4. MODE OF OPERATION

The source format of a digital video signal is usually available in the classical line scanning format. This format is not at all convenient for motion estimation processing. Pixels must be delivered to the STI3220 in the block format which is described hereafter.

Also, when the video signal is a colour video signal, luminance and chrominance pixels are interlaced. Motion estimation is usually performed with the luminance pixels alone, sometimes with the chrominance pixels alone, never with both. This is why in the following, a block of pixels will refer to a block of luminance pixels or a block of chrominance pixels, but not to a block of pixels where chrominance and luminance data is mixed.

Operation of the STI3220 is a succession of sequences of two types: initialization sequences and block sequences. For example, processing one reference block requires an initialization sequence followed by a block sequence. Section 4.4 details the way to combine initialization and block sequences to actually operate the circuit. Sections 4.1 to 4.3 first detail the two types of sequences for different block sizes.

These sequences vary slightly depending on the block height (8 or 16).

During an initialization sequence, for all block sizes, the 15 leftmost columns of the search window are entered into the circuit. No reference block is loaded during this sequence. The duration of an initialization sequence is always 16xM cycles.

During an initialization sequence, the circuit pipeline is initialized with the leftmost part of the search window.

During a block sequence, the N rightmost columns of the search window are loaded into the circuit and the reference block is also loaded into the circuit. The duration of this sequence is always MxN cycles, the time needed to load the reference block. During a block sequence, the distortions relative to the reference block being loaded are computed.

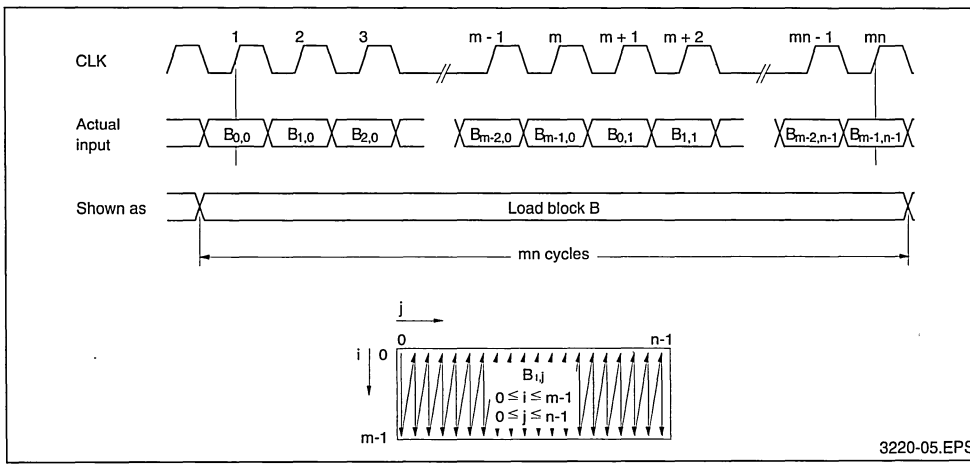
4.1 How to Load a Block

The STI3220 deals with blocks of pixels. A block is a rectangular area of an image and is defined by its width and its height measured in numbers of pixels.

When block B is delivered to the STI3220 (refer to figure 4), it is entered through an 8-bit wide pixel port. The block is scanned column by column, left to right. Each column is in turn scanned from top to bottom. One new pixel of the block must be entered at each clock cycle so that the loading of a mxn sized block will last exactly mxn cycles. Data is latched on the rising edge of CLK. On the rising edge of CLK numbered 1 in figure 4, the first pixel of block B is latched in the circuit.

In all subsequent timing diagrams, whenever a block loading is involved, a simplified representation will be used, as illustrated in Figure 4.

Figure 4 : How to load a block



4.2 8x4n blocks

For 8x4n sized blocks, the search window is 8+15 pixels high and 4n+15 pixels wide. It is divided in 6 sub-windows as shown in Figure 5. They all have the same height which is the reference block height : 8. As the search window height is not a multiple of 8, sub-windows SWC1 and SWC2 extend beyond the search window : their bottom row is outside the search window. These pixels need not be entered in the circuit, but for timing reasons (loading the sub-window SWC1 must last as long as loading sub-window SWB1 or SWA1), some value must be entered in their place. The leftmost sub-windows SWA1, SWB1 and SWC1, are all 15 pixels wide and are loaded during the initialization sequence. The rightmost sub-windows SWA2, SWB2 and SWC2, are all 4n pixels wide and are loaded during the block sequence.

4.2.1 Initialization Sequence

For an illustration of this section refer to Figure 6. The initialization sequence for 8x4n sized reference blocks starts with a $\overline{\text{BSYNC}}$ strobe during the first cycle. During the 8 first cycles, no data is entered in the circuit. During this time, the circuit initialises itself. Then loading of blocks SWA1, SWB1 and SWC1 starts simultaneously through ports A[7:0], B[7:0], C[7:0], respectively. When this loading ends, the sequence is terminated. Its duration is 128 cycles.

Figure 5 : Search Window for 8 x 4n Reference Block

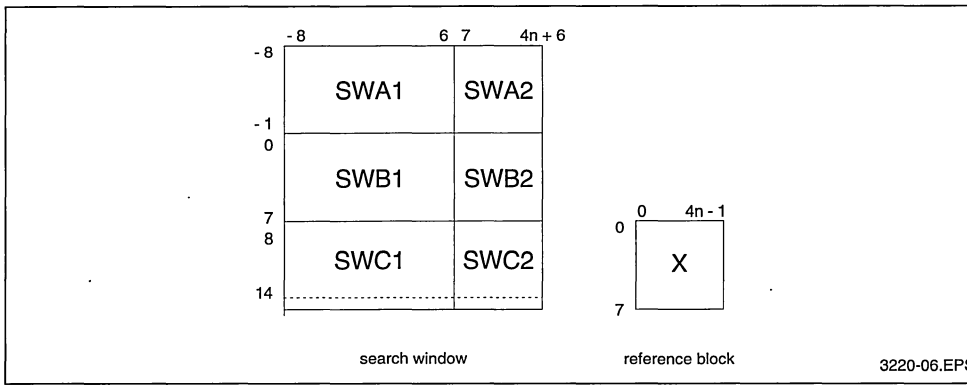
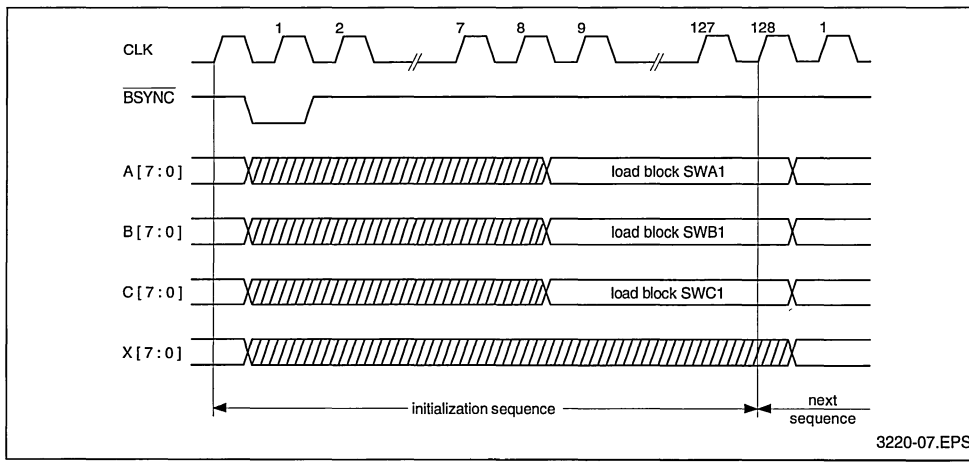


Figure 6 : Initialization Sequence for 8 x 4n Blocks



4.2.2 Block Sequence

For an illustration of this section refer to Figure 7.
When processing rectangular reference blocks ($\overline{SQ}$ high), a $\overline{BSYNC}$ low strobe during the first cycle of every block sequence is compulsory. When processing square reference blocks ($\overline{SQ}$ low), this strobe is optional.During a block sequence, the reference block and sub-windows SWA2, SWB2 and SWC2 are loaded simultaneously through ports X [7 : 0], A [7 : 0], B [7 : 0], C [7 : 0], respectively. This sequence takes exactly 8 x 4n = 32n cycles as all these blocks have the same size.

4.3 16x4n blocks

For 16x4n sized blocks, the search window is 16+15 pixels high and 4n+15 pixels wide. It is divided in 6 sub-windows as shown in Figure 8. They all have the same height which is the reference block height : 16. Sub-windows SWA1, SWA2, SWC1 and SWC2 extend largely beyond the search window. These pixels need not be entered in the circuit, but for timing reasons (loading the sub-windows SWA1 and SWC1 must last as long as loading sub-window SWB1), some value must be entered in their place. An alternative method may be used to enter the required data items without entering unused values. See section 4.3.1.
The leftmost sub-windows SWA1, SWB1 and SWC1, are all 15 pixels wide and are loaded during the initialization sequence. The rightmost sub-windows SWA2, SWB2 and SWC2, are all 4n pixels wide and are loaded during the block sequence.

Figure 7 : Block Sequence for 8 x 4n Blocks

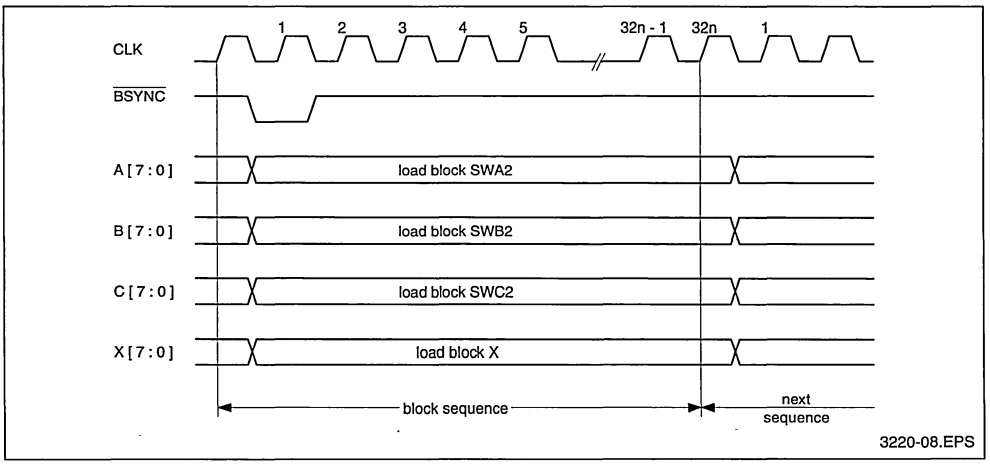
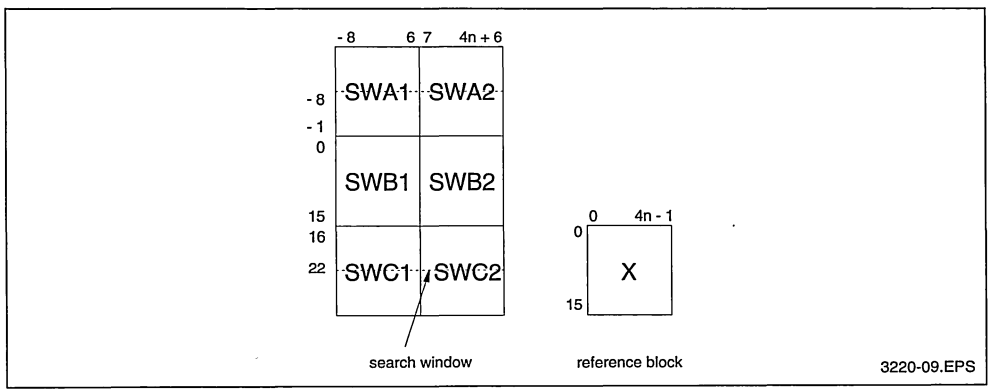


Figure 8 : Search Window for 16 x 4n Reference Blocks



4.3.1 Initialization Sequence

For an illustration of this section refer to Figure 9.
The initialization sequence for 16x4n sized reference blocks starts with a BSYNC strobe during the first cycle. During the 16 first cycles, no data is entered in the circuit. During this time, the circuit initializes itself. Then loading of blocks SWA1, SWB1 and SWC1 starts simultaneously through ports A [7 : 0], B [7 : 0], C [7 : 0], respectively. When this loading ends, the sequence is terminated. Its duration is 256 cycles.
It is worth noting that the useful pixels of block SWC1 (those inside the search window) and the unused pixels of block SWA1 (those outside the search window) are entered in the circuit concurrently, and vice versa. This is because these two blocks have symmetric positions in relation to the search window. Thus, the unused pixels of block SWA1 can be the useful pixels of block SWC1, and vice versa. Practically, this is realised by linking ports A [7 : 0] and C [7 : 0] and writing to this common port a block composed of the 8 upper rows of block SWC1 and the 8 lower rows of block SWA1 (see section 5.3 for a system diagram). However, this composed block still contains unused data : the 8th row of block SWC1, which is outside the search window.

4.3.2 Block Sequence

For an illustration of this section refer to Figure 10.
When processing rectangular reference blocks (SQ high), a BSYNC low strobe during the first cycle of every block sequence is compulsory. When processing square reference blocks (SQ low), this strobe is optional.
During a block sequence, the reference block and sub-windows SWA2, SWB2 and SWC2 are loaded simultaneously through ports X[7:0], A[7:0], B[7:0], C[7:0], respectively. This sequence takes exactly 16x4n=64n cycles as all these blocks have the same size.
The ports A[7:0] and C[7:0] can be linked and the blocks SWC2 and SWA2 partially scanned as seen above to reduce the number of data items delivered to the circuit.

Figure 9 : Initialization Sequence for 16 x 4n Blocks

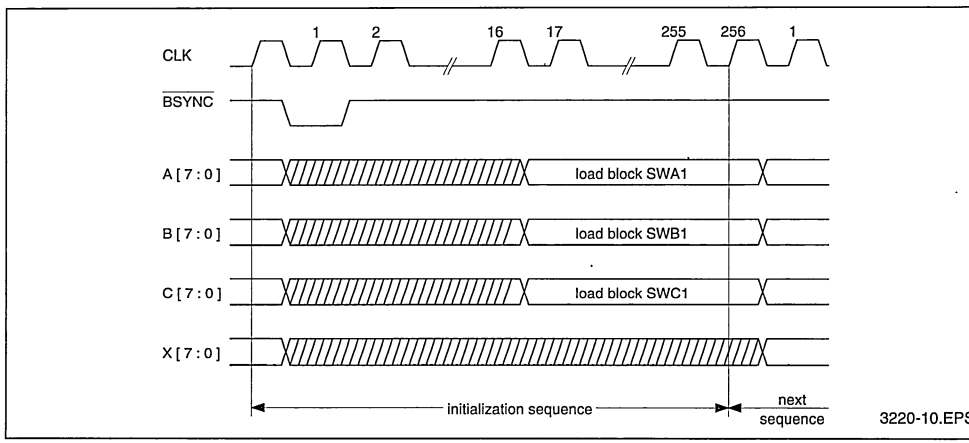
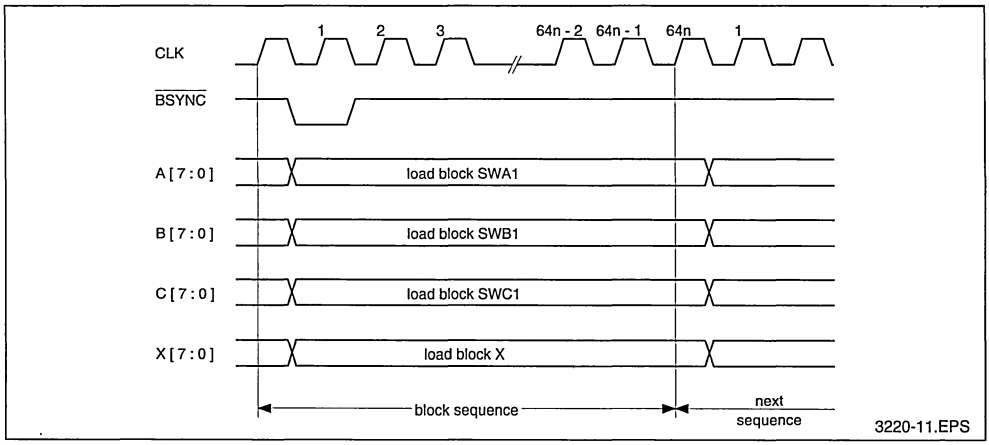


Figure 10 : Block Sequence for 16 x 4n Blocks



4.4 Actual Operation

Actual operation of the circuit is a succession of initialization sequences and block sequences. No dead cycle can be introduced between the successive sequences, as otherwise the circuit pipeline is broken.

4.4.1 Processing One Block

Processing one block requires an initialization sequence immediately followed by a block sequence. During the initialization sequence, the 15 left columns of the search window are entered and stored in the circuit and no computation is performed. During the block sequence, the reference block and the remaining part of the search window are entered and processed in the circuit. During this sequence, the 256 distortions are computed. The output of the minimum distortion and the motion vector takes place a few cycles after the end of the block sequence, see section 4.6 for more details.

4.4.2 Continuous Processing of Blocks

One way to process several blocks is to repeat the above described operations as many times as necessary. However, under certain conditions, there is a more efficient way of processing several blocks. Under these conditions, only one initialization sequence and as many block sequences as there are reference blocks are necessary. The gain is substantial : there is a factor 3 (8x4n blocks) or 2 (16x4n blocks) gain in memory bandwidth and operating frequency.

The only condition to be observed in using this mode is that the 15 rightmost columns of search window k are exactly the 15 leftmost columns of search window k+1. The reason why this mode is more efficient is easily understandable. The circuit takes advantage of the fact that successive search windows overlap. At the end of block sequence k, the 15 leftmost columns of search window k+1 are already loaded in the circuit so that no initialization sequence is required for block k+1.

The first and unique initialization sequence loads in the 15 leftmost columns of the first search window. It is immediately followed by the successive block sequences. During block sequence k, results relative to block sequence k-1 are accessible through the I/O port.

In particular, continous processing of blocks is possible when reference block k is the right neighbour of reference block k-1. In this case the condition that two successive search windows overlap by 15 columns is met.

If the edge control is set correctly, the pipeline is not broken when a search window is situated near the right edge of a frame and does not therefore overlap with the next search window. For more details on how to do this, see section 5.

4.5 The Block Synchronisation signal : BSYNC

An initialisation sequence always starts with a BSYNC strobe (active low). This sets the circuit ready for operation.

When rectangular blocks are processed (SQ high), the circuit must be informed when a new block starts. This is done by strobing BSYNC low during the first cycle of every block sequence.
When square blocks are processed (SQ low), the circuit knows the size of the block so that the BSYNC strobe is optional during block sequences. Signal H8 and SQ are read in the circuit at each BSYNC strobe.

4.6 Output of Results

Refer to Figure 11.
At the beginning of every sequence, the circuit computes the minimum and its address among the values stored in the distortion RAM. If the previous sequence was a block sequence, the circuit computes the motion vector and the minimum distortion relative to the reference block entered during this sequence. If it was an initialization sequence, the results are meaningless.
When the results are available, the MVSYNC signal goes low. This happens during the (M+30)th cycle of every sequence (M=8 or 16). What happens next depends on the state of the IOB bus.

If OE is high, the motion vector is present on the IOB bus during one cycle, while MVSYNC is low. Then MVSYNC goes high on the next cycle, during which nothing happens. Then DSYNC0 goes low while the minimum distortion msb is present on IOB. Then DSYNC0 goes high again and DSYNC1 goes low while the minimum distortion lsb is present on IOB. If OE goes low while the results are outputted, the data not yet delivered is lost.
If OE is low, IOB is in the high impedance state and MVSYNC remains low. When OE goes high again the data is outputted as described above. This feature is useful when several STI3220 circuits are used in parallel : the circuits can share a common bus and although all the results are ready at the same time, the system can read them sequentially. If OE does not go high before a new sequence starts, then MVSYNC returns to a high state and the data is lost.
If, in a given distortion set, several distortions are equal to the minimum of the set, the circuit will output the motion vector which has the highest priority according to the following table.

	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-8	2	12	30	58	90	122	154	186	187	155	123	91	59	31	13	3
-7	0	6	20	42	74	106	138	170	171	139	107	75	43	21	7	1
-6	8	22	44	76	108	140	172	202	203	173	141	109	77	45	23	9
-5	4	14	32	60	92	124	156	188	189	157	125	93	61	33	15	5
-4	18	40	72	104	136	168	200	226	227	201	169	137	105	73	41	19
-3	10	28	56	88	120	152	184	214	215	185	153	121	89	57	29	11
-2	26	54	86	118	150	182	212	236	237	213	183	151	119	87	55	27
-1	16	38	70	102	134	166	198	224	225	199	167	135	103	71	39	17
0	36	68	100	132	164	196	222	244	255	223	197	165	133	101	69	37
1	24	52	84	116	148	180	210	234	235	211	181	149	117	85	53	25
2	50	82	114	146	178	208	232	249	250	233	209	179	147	115	83	51
3	34	66	98	130	162	194	220	242	243	221	195	163	131	99	67	35
4	64	96	128	160	192	218	240	253	254	241	219	193	161	129	97	65
5	48	80	112	144	176	206	230	247	248	231	207	177	145	113	81	49
6	62	94	126	158	190	216	238	251	252	239	217	191	159	127	95	63
7	46	78	110	142	174	204	228	245	246	229	205	175	143	111	79	47

3220-03.TBL

There is no simple algorithm which can describe this table, it is just a consequence of the circuit architecture. Note, however, that the (0,0) motion vector has the highest priority.
The following table gives the exact bit allocation of bus IOB when results are delivered. Vj[3:0] is the horizontal component of the motion vector and is coded in 2s complement. Similarly, Vi[3:0] is the vertical component of the motion vector. The minimum distortion is an unsigned value.

	IOB7	IOB6	IOB5	IOB4	IOB3	IOB2	IOB1	IOB0
Motion Vector	Vj3	Vj2	Vj1	Vj0	Vi3	Vi2	Vi1	Vi0
Distortion MSB	D15	D14	D13	D12	D11	D10	D9	D8
Distortion LSB	D7	D6	D5	D4	D3	D2	D1	D0

4.7 How to Read a Distortion

When a minimum computation is not in process, the distortion RAM can be read by the system through the IOB bus. This particular mode of operation is illustrated in Figure 12.
The address must be entered while the bus is in high impedance state (OE low). Then OE must go high in order to let the circuit write on the bus, (OE signal must change value while CLK is high). The address taken into account is the one present on the bus during the last rising edge of CLK when OE was low. One dead cycle elapses, and the distortion is written out, first the msbs then the lsbs. Four cycles are thus necessary to read a distortion.
The address of a distortion is the related motion vector. The bit allocation is exactly the same as seen above in section 4.6.
When a minimum computation is in process, the distorsion RAM cannot be read.
M + 1 cycles after the beginning of every sequence, a new set of distorsions is stored in the RAM. These distortions are those related to the previous sequence. A minimum computation immediately follows and lasts 28 cycles as shown in Figure 13.
Then the results are delivered and this takes another 4 cycles. The RAM is then available and distortions of the new set can be read until another set is stored in RAM, M + 1 cycles after the beginning of the next sequence.

Figure 11 : Output of Results

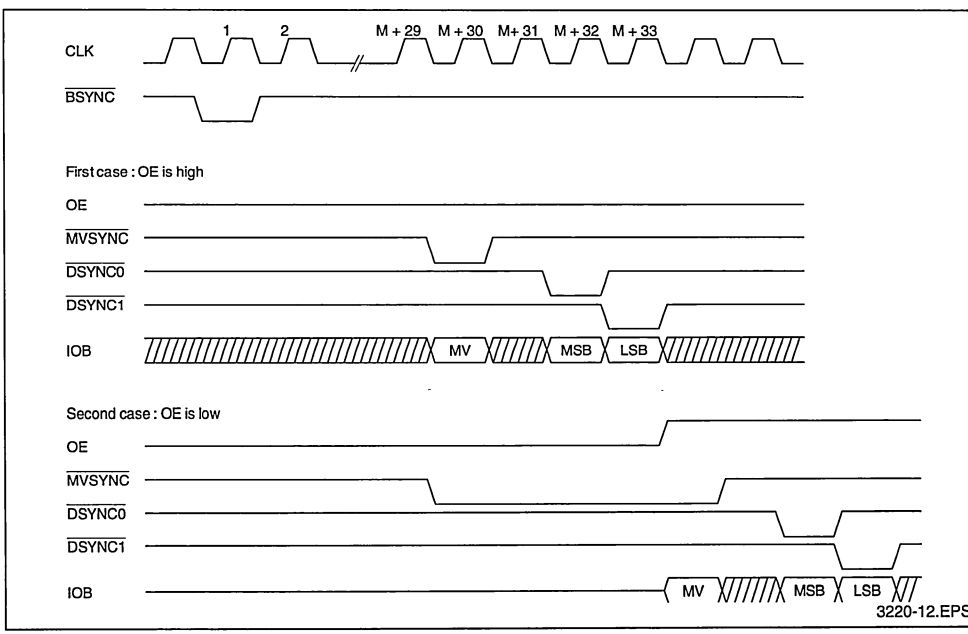


Figure 12 : Reading a Distortion

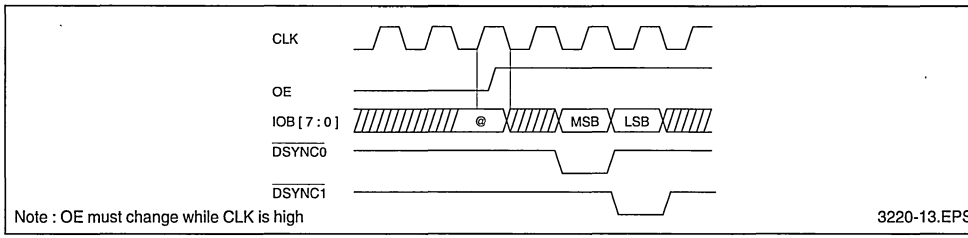
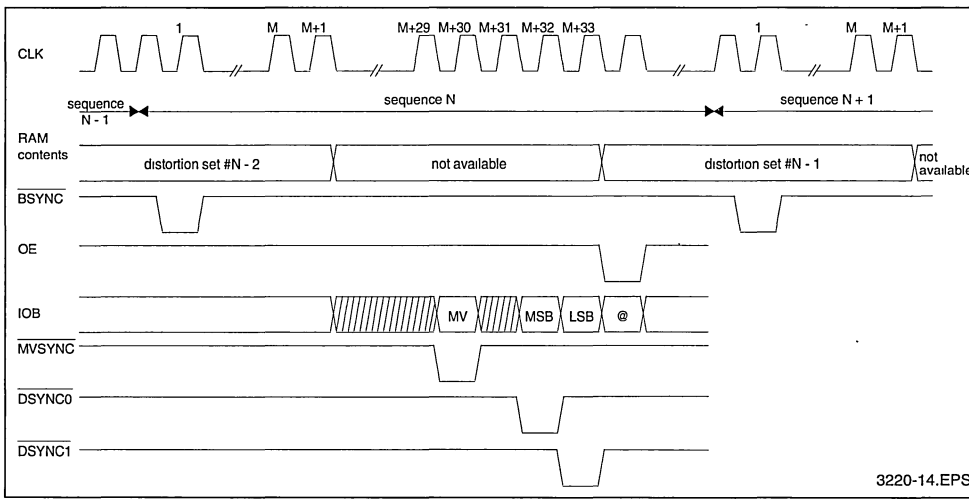


Figure 13 : RAM Availability



4.8 How to Use the Edge Control

When a search window is partially outside the actual frame, the number of candidate blocks is less than 256 and the set of valid motion vectors is reduced. The STI3220 can take this into account and compute motion vectors in the valid range. There are 9 different cases depending on whether the search window is along one of the edges or in one of the corners of a frame.

To set the edge control, four pins are provided. They are all latched in the circuit during the first cycle of every block sequence and are related to the search window loaded during this very sequence.

Each pin is related to one edge of the frame (left, right, top or bottom) and setting one pin high during the first cycle of a block sequence will mean that the current search window (the one being loaded into the circuit during this particular block sequence) extends beyond the edge of the frame related to that pin, and that the valid set of motion vectors for the current search window is reduced. The most common case is when the search window is entirely included in the frame, and the four pins are set low.

Pin EC0 is related to the top edge of the frame, and setting this pin high will reduce the valid range of motion vectors to the ones with a positive or null vertical component.

Pin EC1 is related to the right edge of the frame, and setting this pin high will reduce the valid range of motion vectors to the ones with a negative or null horizontal component.

Pin EC2 is related to the bottom edge of the frame, and setting this pin high will reduce the valid range of motion vectors to the ones with a negative or null vertical component.

Pin EC3 is related to the left edge of the frame, and setting this pin high will reduce the valid range of motion vectors to the ones with a positive or null

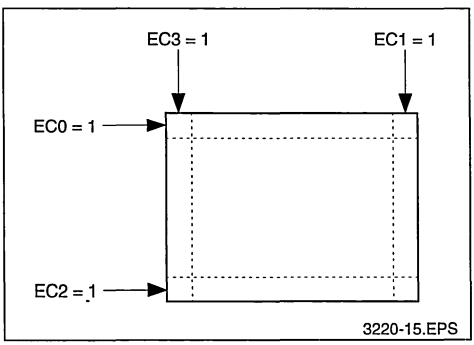
horizontal component.
When the search window extends beyond a corner of the frame, two edges have to be selected.
Unrealistic combinations of edges are not prohibited and will yield logical results. For example, selecting the top **and** the bottom edge will reduce the valid set of motion vectors to the horizontal ones. All combinations are summarized in the following table. Unrealistic combinations are indicated with a*.

				Valid Range of Motion Vectors	
EC3	EC2	EC1	EC0	Horizontal Component	Vertical Component
0	0	0	0	$-8 \le j \le 7$	$-8 \le i \le 7$
0	0	0	1	$-8 \le j \le 7$	$0 \le i \le 7$
0	0	1	0	$-8 \le j \le 0$	$-8 \le i \le 7$
0	0	1	1	$-8 \le j \le 0$	$0 \le i \le 7$
0	1	0	0	$-8 \le j \le 7$	$-8 \le i \le 0$
0*	1	0	1	$-8 \le j \le 7$	$i = 0$
0	1	1	0	$-8 \le j \le 0$	$-8 \le i \le 0$
0*	1	1	1	$-8 \le j \le 0$	$i = 0$
1	0	0	0	$0 \le j \le 7$	$-8 \le i \le 7$
1	0	0	1	$0 \le j \le 7$	$0 \le i \le 7$
1*	0	1	0	$j = 0$	$-8 \le i \le 7$
1*	0	1	1	$j = 0$	$0 \le i \le 7$
1	1	0	0	$0 \le j \le 7$	$-8 \le i \le 0$
1*	1	0	1	$0 \le j \le 7$	$i = 0$
1*	1	1	0	$j = 0$	$-8 \le i \le 0$
1*	1	1	1	$j = 0$	$i = 0$

3220-05.TBL

$EC_n = 0$ unless otherwise noted.

Figure 14



4.9 How to Set the Block Size : signals $\overline{SQ}$ and $\overline{H8}$

To set the block size, two pins are provided. Both are latched in when $\overline{BSYNC}$ is strobed low.
To process 8 pixel high reference blocks, signal $\overline{H8}$ must be low during the $\overline{BSYNC}$ strobe. To process 16 pixel high reference blocks, signal $\overline{H8}$ must be high during the $\overline{BSYNC}$ strobe.

To process square reference blocks, signal $\overline{SQ}$ must be low during the $\overline{BSYNC}$ strobe. To process rectangular reference blocks, signal $\overline{SQ}$ must be high during the $\overline{BSYNC}$ strobe.
The following table summarizes the four different block sizes:

$\overline{H8}$	$\overline{SQ}$	Block Size
0	0	8 x 8
0	1	8 x 4n
1	0	16 x 16
1	1	16 x 4n

When the circuit is set to process rectangular blocks, it does not know the width of the block and so must be told when a new block starts. This is why, when rectangular blocks are processed, a $\overline{BSYNC}$ strobe is compulsory during the first cycle of every block sequence. It may be noted, for reference purposes only, that in this mode the successive reference blocks need not have the same width, provided their width is a multiple of 4. When the circuit is set to process square blocks, $\overline{BSYNC}$ strobes are optional at the beginning of every block sequence. They are required only with

initialization sequences. If the circuit is used in pipeline mode as explained in section 4.4.2, then only one BSYNC strobe is needed during the unique initialization sequence.

4.10 How to Reset the circuit : signal RESET

The STI3220 must be reset before operation. This is done by strobing the signal RESET low during at least one cycle before any initialization sequence starts. The clock must be running.

4.11 How to measure the circuit latency time : signal OSYNC

The circuit latency time Tlat is the time elapsing between the input of the first pixel of a reference block, and the output of the motion vector relative to this block.

The motion vector is ready a few cycles after the last pixel of the reference block is entered in the circuit, 38 cycles for 8x4n sized reference blocks and 46 cycles for 16x4n sized reference blocks. The latency time is thus 32n+38 cycles for 8 x 4n sized reference blocks and 64n+46 cycles for 16 x 4n sized reference blocks.

To ease system implementation when this latency time must be taken into account, e.g. for the initialization of circuits expecting results from the STI3220, the OSYNC signal is provided.

The circuit generates one OSYNC low strobe for every BSYNC strobe. This occurs exactly Tlat cycles after the related BSYNC strobe, concurrently with the MVSYNC strobe indicating that the motion vector is ready.

4.12 How to freeze the circuit : signal EN

It may be useful to freeze the circuit in an asynchronous way for an indeterminate period of time without breaking the circuit pipeline or loosing any data. This is possible with the STI3220 because it is a fully static CMOS device.

When signal EN is high, the circuit internal clock is frozen until signal EN goes low again.

To ensure no loss of data during the transitions between the frozen and the working states of the circuit, the EN signal must fulfill the requirements specified in section 6.

5 - BASIC STI3220 APPLICATION EXAMPLES

5.1. - Picture borders : example with 8*8 blocks, displacement range -8 to +7:

When computing a block in the middle of a picture in pipeline mode, the left and middle parts of the search window (Le and Mi columns) are already loaded into the STI3220 and the right part (Ri columns) is input with the reference block. The position of that right area in the picture is shifted 8 columns right from the reference block position (see Figure 15).

However when the reference block is the last one of a row of blocks in the image, only one row for the search window is available on the right of this reference block. In order not to break the pipeline mode, the remaining 7 columns of the search window will be the 7 columns corresponding to the next block that will be evaluated in the pipeline i.e. the first block position of the next row of block (see Figure 16). Those 7 columns are loaded into the chip but will not be taken into account for the motion vector calculation if the Edge Control signals are set to EC3...EC0 = 0010 (right hand edge).

The first reference block of the next line is entered in the chip in synchronism with search window columns from 7 to 15 (see Figure 17). The middle part of the search window has already been loaded during the previous block and the left part of the search window is not taken into account for motion vector calculation by setting the Edge Control bits to EC3...EC0 = 1000 (left hand edge).

Of course the same principle can be applied when reaching the end of an image (see Figure 18): the search window corresponding to the first block position of an image is loaded during the processing of the last block of an image : pipeline processing is never broken. Just notice in that case that the first column inputted is only significant on the A and B bands (last column of the image) and that the 7 following columns are only significant on B and C bands (first columns of the image).

The pipeline continuity on the borders of an image is always respected whatever the blocks' size is.

Figure 15

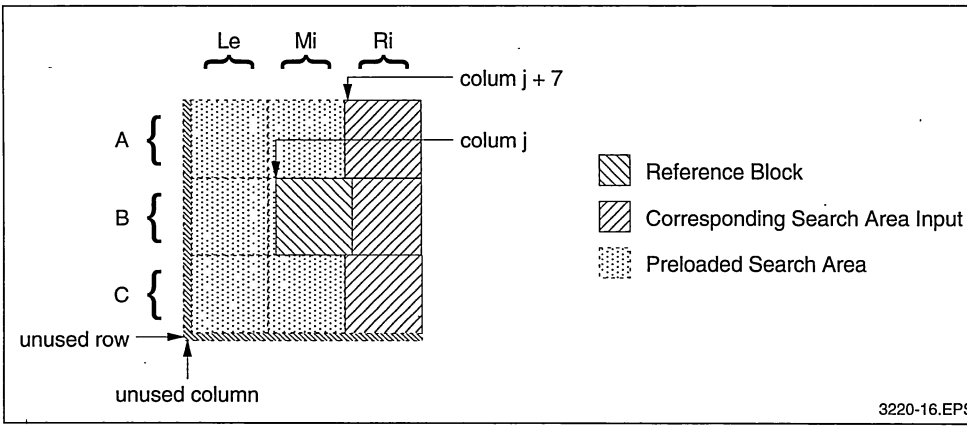


Figure 16

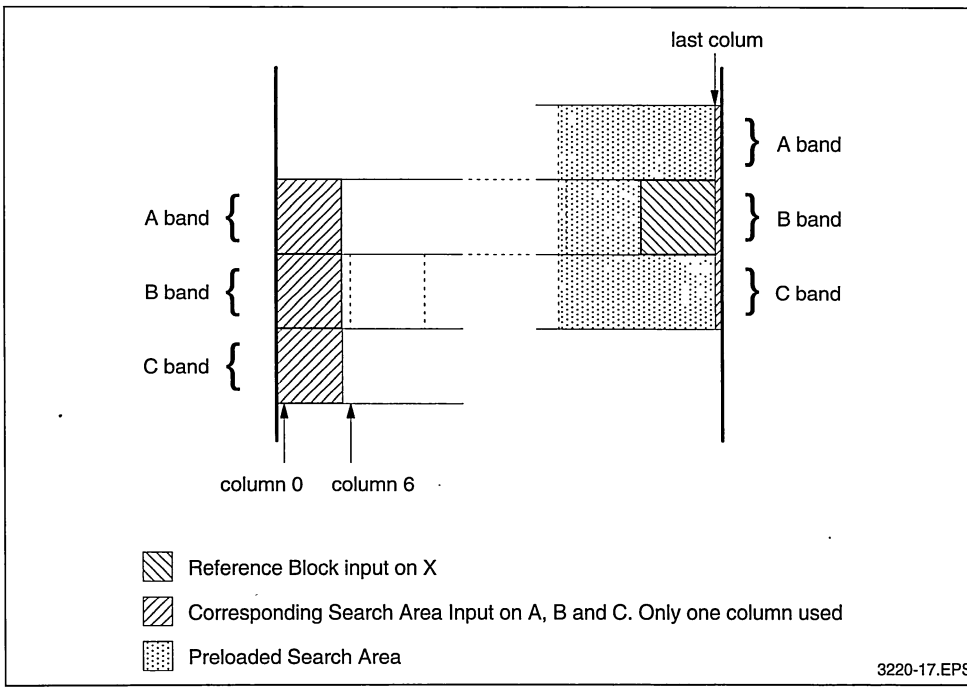


Figure 17

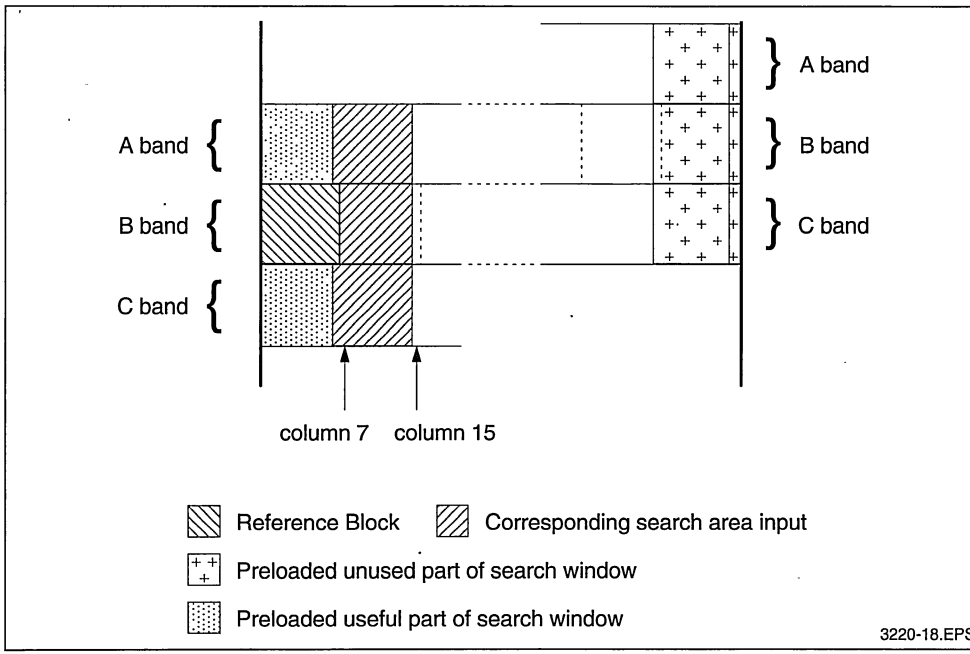
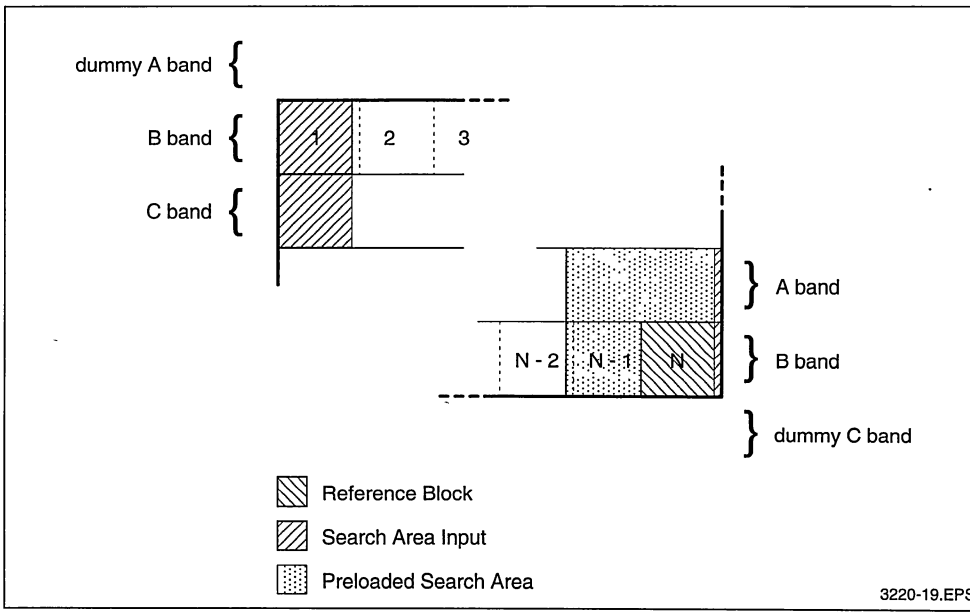


Figure 18



5.2. - Example of system outline in 8 x 4n mode:

use of delay lines: in synchronism with the reference block, the frame memory is accessed on the line of blocks under the reference one. Two delay lines are necessary to provide the STI 3220 with the middle and upper band of the search window. For low rate applications it may be cheaper to suppress the delay lines and to access 3 consecutive times into the frame memory for each pixel of the reference block (see Figure 20).

Figure 19

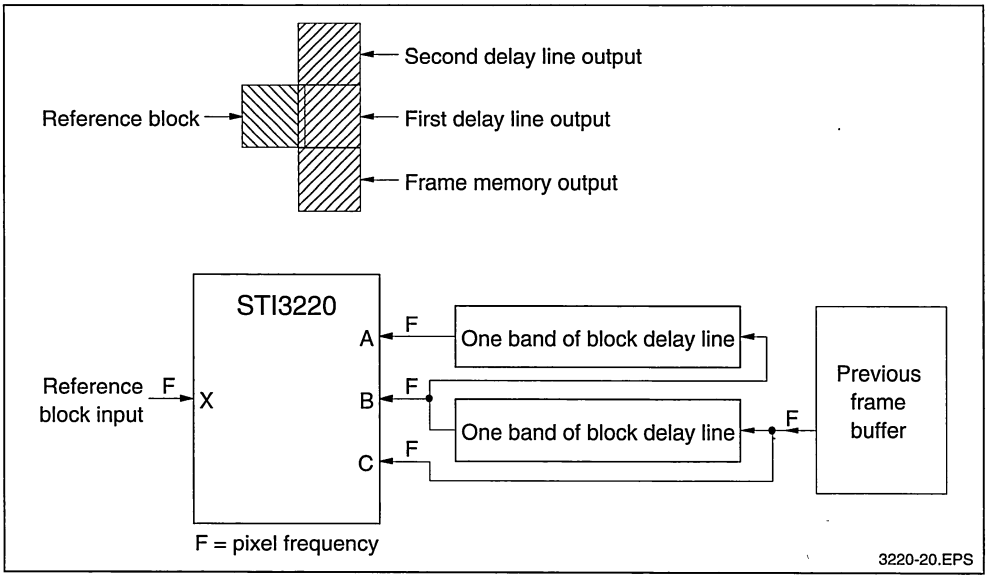
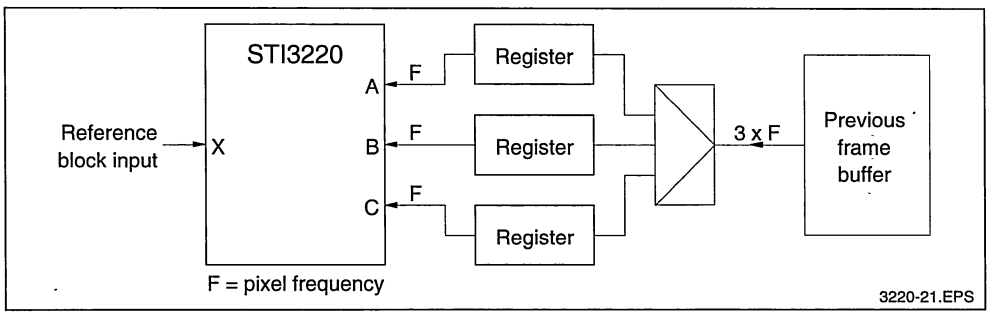


Figure 20



5.3. - Example of system outline in 16*16 mode:

In this case the search window is only one half block high above the reference block (8-pixels high) and one half block high under (in fact 7-pixels high). The total height of the search window is only two blocks high. When inputting the 8 first pixels of a column of the reference block, only the 8 corresponding pixels of the search window on band B and C are taken into account. In the same way, when inputting the 8 last pixels of a column of the reference block, only the 8 corresponding pixels of the A and B bands are taken into account.

Figure 21

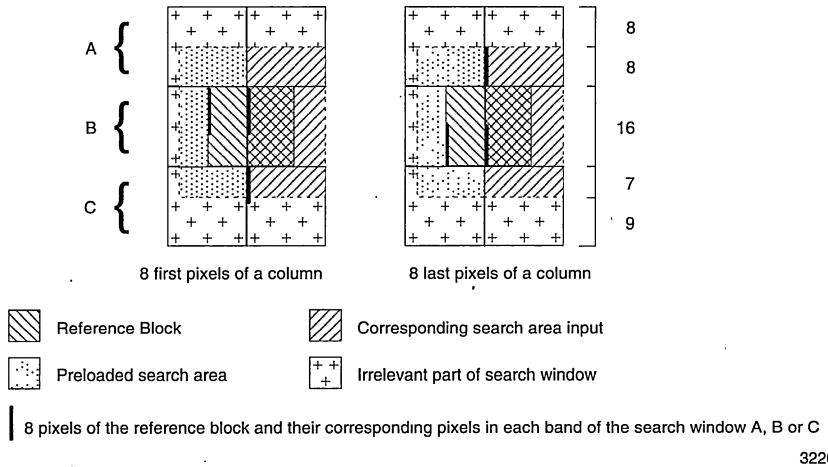
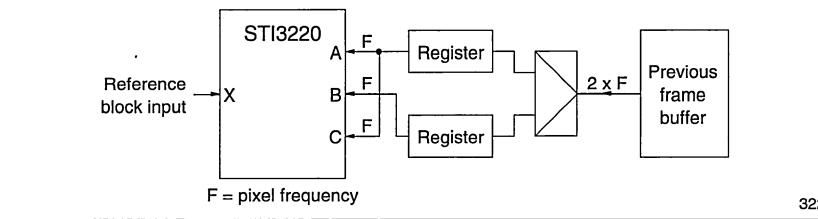


Figure 22



A system architecture using a band by band scanning and 3 delay lines may be implemented in this case (refer to 8*4n example of system outline). For low rate applications it is possible to access only twice the pixel rate on the frame memory in order to provide the search window : as bands A and C are not read at the same time by the chip, they can be connected together (see Figure 22).

5.4 - principle of +15/-15 research (16*16 blocks) :

Computing +15/-15 displacements with the STI 3220 chip able to compute +7/-8 displacements is made possible by dividing the total search window into 4 sub search windows (see Figure 23) : 4 corresponding motion vectors and minimum distorsions will be calculated and the system must manage those results in order to decide what will be the final motion vector for all the search window. The four partial motion vectors can be computed in two ways:

- using 4 STI 3220 chips each one loaded at the same time with the same reference block and one specific search window per chip. The computation costs 2 blocks (1 initialization sequence + 1 block sequence).
- using only one STI 3220 chip loaded 4 consecutive times with the same reference block and each of the specific search window. As the recovery between the different sub search windows is not a multiple of 16, the pipeline mode cannot be used and the computation of the +15/-15 displacement costs 8 blocks 16*16 (4 initialization sequences + 4 block sequences):
  - phase 1 : initialization sequence of sub-window 1 (first 15 columns). Input of a dummy reference block.
  - phase 2 : block sequence of sub-window 1 : input of the 16 last columns of the sub-window 1 and the reference block.
  - phase 3 : initialization sequence of sub-window 2. Result of the first sub-window (motion

vector + minimum distortion) obtained after the 46th cycle of that phase.
- phase 4 : block sequence of sub-window 2.
- phase 5 : initialization sequence of sub-window 3. Result of sub-window 2 after the 46th cycle.
- phase 6 : block sequence of sub-window 3.
- phase 7 : initialization sequence of sub-window 4. Result of sub-window 3 after the 46th cycle.
- phase 8 : block sequence of sub-window 4.

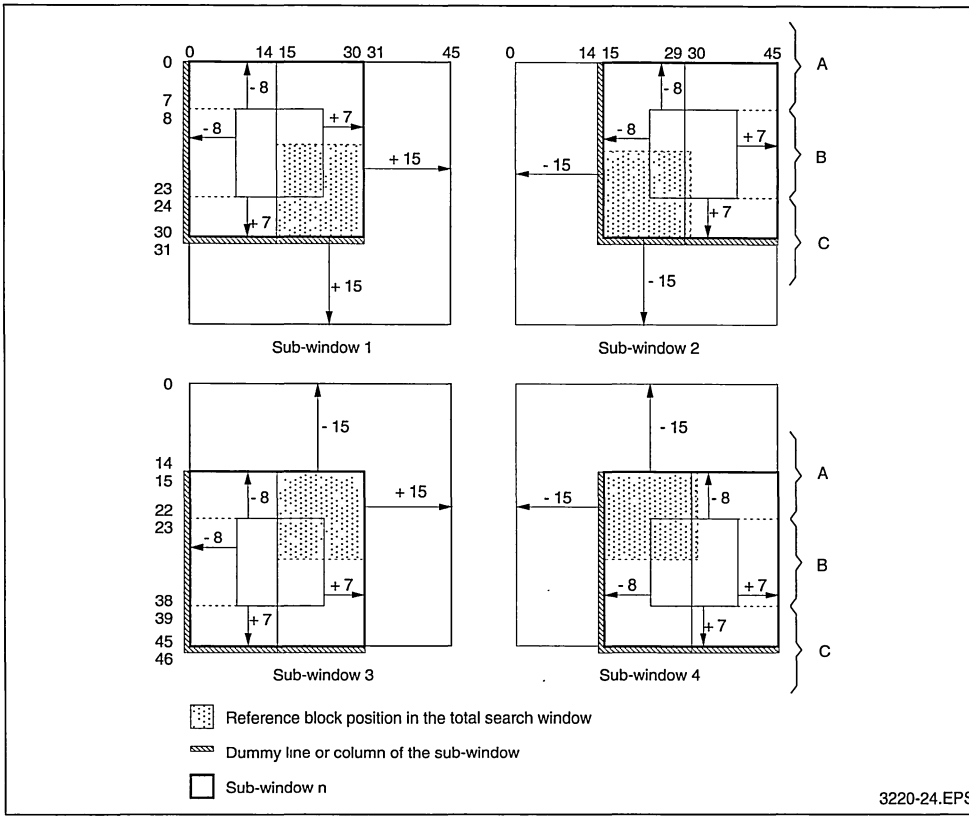
The result for sub-window 4 is obtained after the next 46 cycles, and hence will typically be available during the beginning of the computations for the next reference block.

It is possible in this way to compute the displacement for a CSIF format picture as defined by the CCITT : 352 x 288 pixels x 15 Hz = 1.52 Mpixels/s.

Eight accesses to the chip for one input pixel means a chip working frequency equal to 12.16 MHz.

Note: Another alternative method of computing a -/+ 15 motion vector is to carry out the calculations on a -16/+15 search window (i.e. two times -8/+7 in both directions) and to clip the motion vector to -15/+15 range (i.e. if motion vector is equal to -16 it is forced to -15). The advantage of using a -16/+15 search window is that the block sequence of sub-window 1 is identical to the initialization sequence of sub-window 2 (idem for sub-windows 3 and 4). That means that the pipeline mode can be used between sub-windows 1 and 2, and sub-windows 3 and 4. The total computation time of the -16/+15 motion vector then costs 6 blocks (2 initialization sequences + 4 block sequences) instead of 8 blocks.

Figure 23



6. ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{CC}	Supply Voltage	6	V
T_{OPER}	Operating Temperature Range	0, 70	°C
	Voltage on any pin relative to V_{SS}	6	V

3220-06.TBL

DC ELECTRICAL CHARACTERISTICS

Operating conditions : V_{SS} = 0V, T_A = 0 to 70°C, V_{CC} = 5V ± 5%, unless otherwise specified

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
V_{CC}	Operating Voltage		4.75		5.25	V
I_{CC}	Supply Current • F_{CLK} = 18MHz • F_{CLK} = 0MHz	C_{LOAD} = 50pF (all outputs) All inputs at V_{CC} or V_{SS}			400 1	mA
V_{IL} V_{IH}	Input Voltage Level • Logic Low • Logic High	V_{CC} = 5 ± 0.25V	 2		0.8	V
	High Impedance Input Leakage • I/O Buffers • Input Buffers	V_{IN} = VSS to VCC	 -5 -1		 +5 +1	µA
V_{OL} V_{OH}	Output Voltage Level • Logic Low I_{LOAD} = 500µA • Logic High I_{LOAD} = - 500µA	V_{CC} = 4.75V	 2.7		0.4	V
	Clock Input Voltage Level • Logic Low • Logic High	V_{CC} = 5 ± 0.25V	 2.5		0.6	V
	Input Capacitance	V_{offset} = 2.5V, F = 1MHz			10	pF

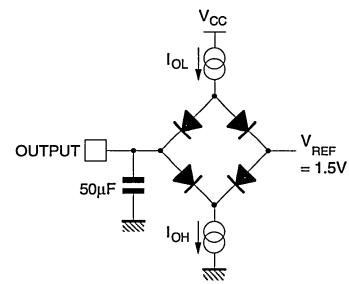
3220-07.TBL

AC ELECTRICAL CHARACTERISTICS

Operating conditions : V_{SS} = 0V, T_A = 0 to 70°C, V_{CC} = 5V ± 5%, unless otherwise specified

Output Loads : Capacitance = 50pF, Current Logic Low = 500µA

TEST LOAD ON OUTPUTS



3220-25.EPS

7. TIMING DIAGRAMS

TIMING PARAMETERS

Symbol	Parameter	Min.	Typ.	Max.	Unit
t_{CH}	Clock Pulse High Width	20			ns
t_{CL}	Clock Pulse Low Width	20			ns
t_{CLK}	Clock Period	50			ns
t_R	Clock Rise Time (see note)	0		10	ns
t_F	Clock Fall Time (see note)	0		10	ns
t_{SDCL}	Data Setup Time from CLK ↑	8			ns
t_{HDCL}	Data Hold Time from CLK ↑	0			ns
t_{DO}	Output Dta Delay from CLK ↑			20	ns
t_{EN1}	Enable Hold Time from CLK ↑	0			ns
t_{EN2}	Enable Rising Edge Setup Time from CLK ↓	5			ns
t_{EN3}	Enable Falling Edge Setup Time from CLK ↑	30			ns
t_{OFF}	Delay from OE ↓ to Output going to High Impedance			20	ns
t_{ON}	Delay from OE ↑ to Output going to High Impedance			20	ns
t_{OE1}	CLK Rising Edge to OE going Low or High	0			ns
t_{OE2}	OE Setup Time	8			ns

3220-08.TBL

Note : The clock edges should be monotic between V_{IL} and V_{IH}.

TIMING WAVEFORMS

Figure 24 : Clock Timing Waveform

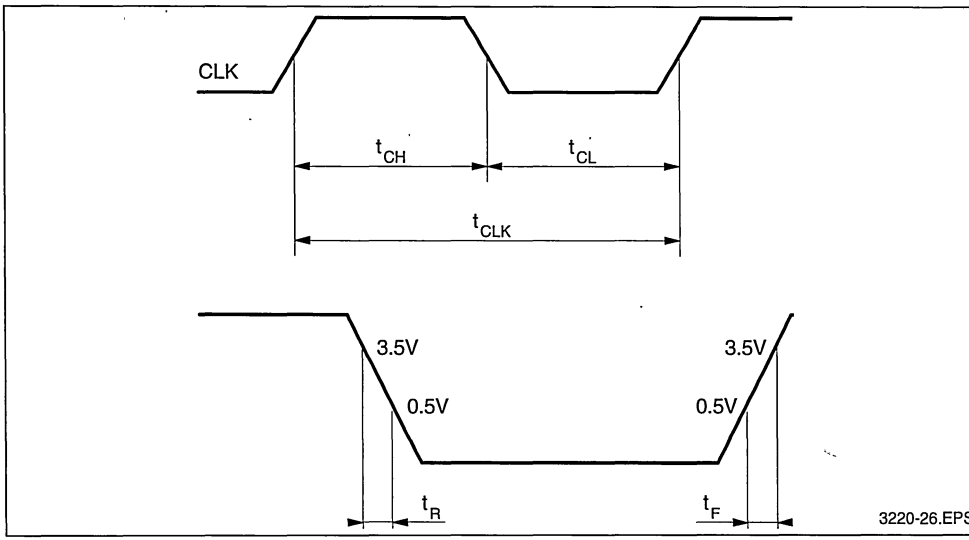


Figure 25 : Data Timing Waveforms

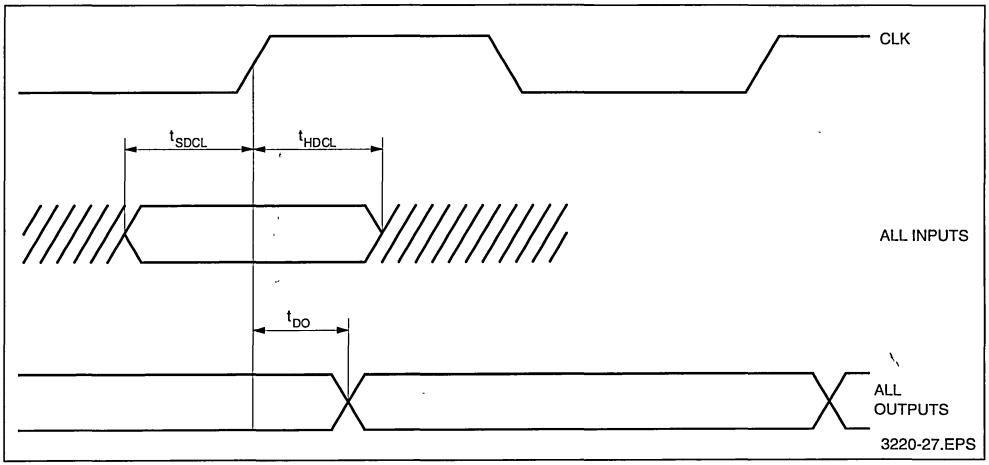


Figure 26 : Clock Enable Timing Waveform

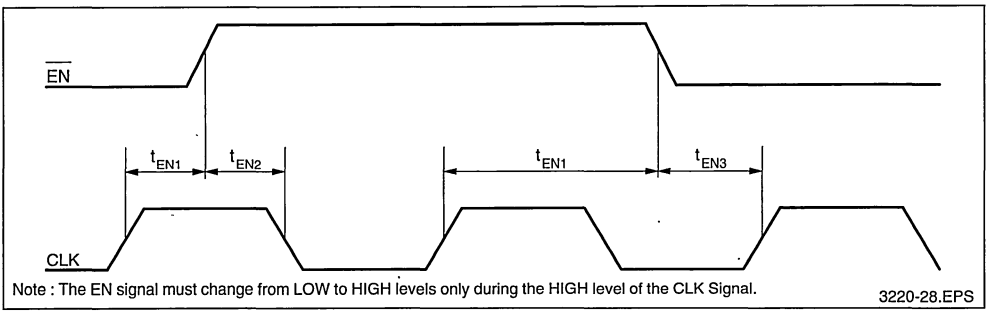
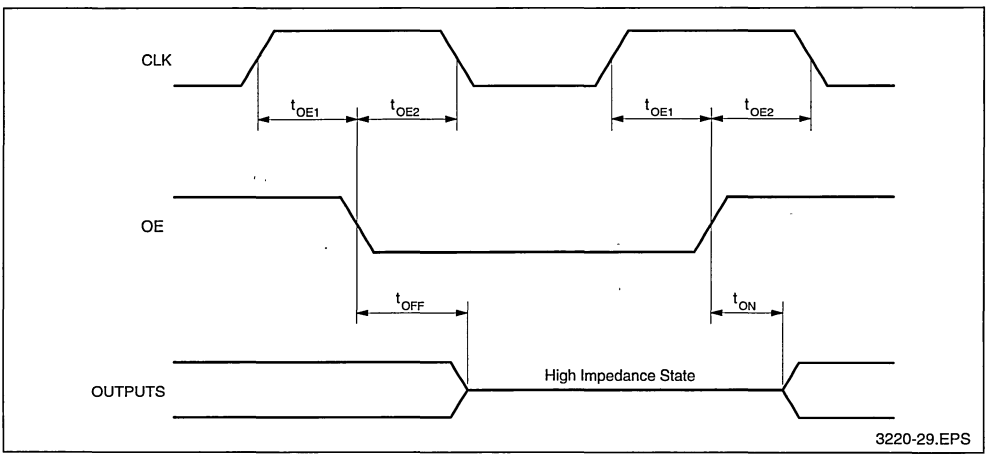
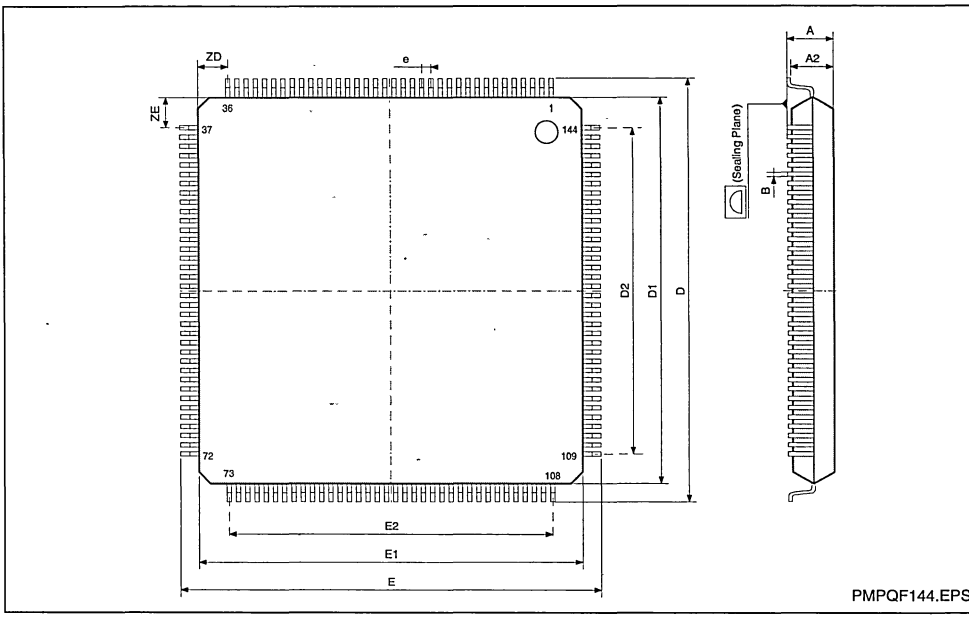


Figure 27 : Output Enable Timing Waveform



PACKAGE MECHANICAL DATA

144 PINS - PLASTIC QUAD FLAT PACK



Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A			3.92			.160
A2	3.17	3.42	3.67	0.125	0.134	.144
B						
D	30.95	31.20	31.45	1.219	1.228	1.238
D1	27.90	28.00	28.10	1.098	1.102	1.106
D2		22.75			0.896	
e		0.65			0.026	
E	30.95	31.20	31.45	1.219	1.228	1.238
E1	27.90	28.00	28.10	1.098	1.102	1.106
E2		22.75			0.896	
ZD		2.63			0.104	
ZE		2.63			0.104	

PQFP144.TBL

DATASHEETS

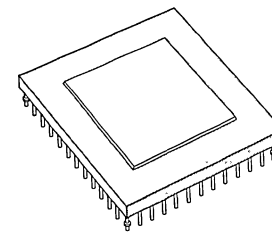
IMAGE PRE-POST PROCESSING DEVICES

SGS-THOMSON MICROELECTRONICS

IMSA110

IMAGE AND SIGNAL PROCESSING SUB–SYSTEM

- 1-D/2-D SOFTWARE CONFIGURABLE CONVOLVER/FILTER
- ON-CHIP PROGRAMMABLE LINE DELAYS (0 — 1120 STAGES)
- 8-BIT DATA AND 8.5-BIT COEFFICIENT SLICE
- 21 MULTIPLY-AND-ACCUMULATE STAGES
- 1-D (21) OR 2-D (3 x 7) CONVOLUTION WINDOW
- ON-CHIP POST PROCESSOR FOR DATA TRANSFORMATION
- FULLY CASCADABLE IN WINDOW SIZE AND ACCURACY
- 20 MHZ DATA THROUGHPUT (420 MOPS)
- SIGNED/UNSIGNED DATA AND COEFFICIENTS
- MICROPROCESSOR INTERFACE
- HIGH SPEED CMOS IMPLEMENTATION
- TTL COMPATIBLE
- SINGLE +5V ± 10% SUPPLY
- POWER DISSIPATION < 2.0 WATTS
- 100 PIN CERAMIC PGA



PGA100
(Ceramic Grid Array Package)

APPLICATIONS

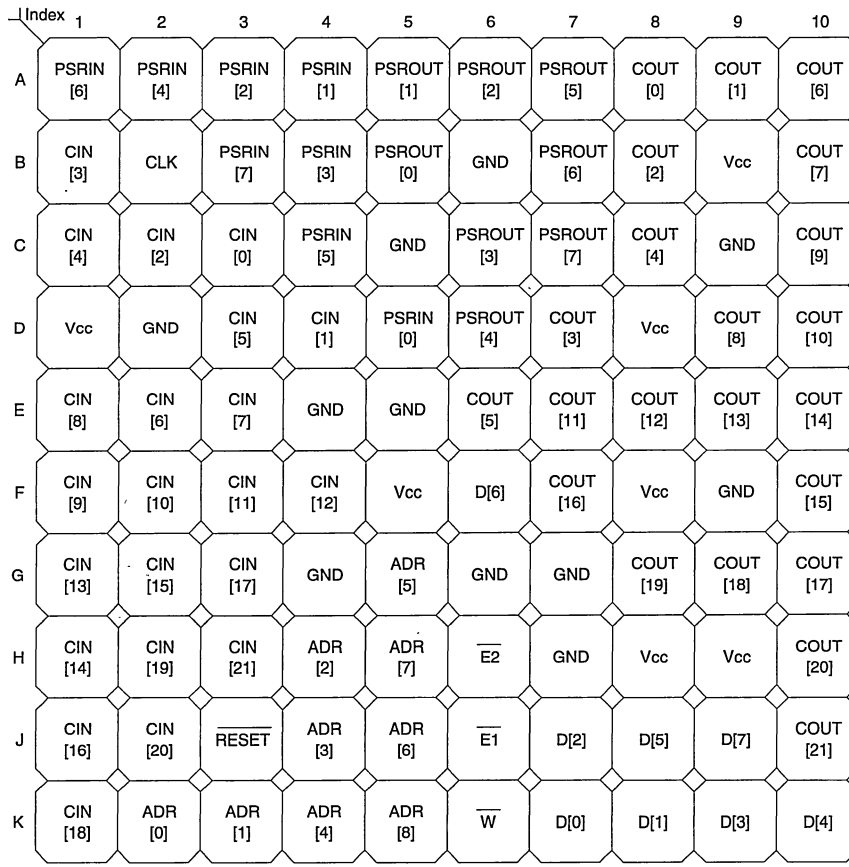
- 1-D and 2-D digital convolution and correlation
- Real time image processing and enhancement
- Edge and feature detection
- Data transformation and histogram equalisation
- Computer vision and robotics
- Template matching
- Pulse compression
- 1-D or 2-D interpolation

ORDERING INFORMATION

Part Number	Package	Clock Speed	Military/ commercial
IMSA110-G20S	PGA100	20MHz	commercial

A110-01.TBL

PIN CONNECTIONS



Index	1	2	3	4	5	6	7	8	9	10
A	PSRIN [6]	PSRIN [4]	PSRIN [2]	PSRIN [1]	PSROUT [1]	PSROUT [2]	PSROUT [5]	COUT [0]	COUT [1]	COUT [6]
B	CIN [3]	CLK	PSRIN [7]	PSRIN [3]	PSROUT [0]	GND	PSROUT [6]	COUT [2]	Vcc	COUT [7]
C	CIN [4]	CIN [2]	CIN [0]	PSRIN [5]	GND	PSROUT [3]	PSROUT [7]	COUT [4]	GND	COUT [9]
D	Vcc	GND	CIN [5]	CIN [1]	PSRIN [0]	PSROUT [4]	COUT [3]	Vcc	COUT [8]	COUT [10]
E	CIN [8]	CIN [6]	CIN [7]	GND	GND	COUT [5]	COUT [11]	COUT [12]	COUT [13]	COUT [14]
F	CIN [9]	CIN [10]	CIN [11]	CIN [12]	Vcc	D[6]	COUT [16]	Vcc	GND	COUT [15]
G	CIN [13]	CIN [15]	CIN [17]	GND	ADR [5]	GND	GND	COUT [19]	COUT [18]	COUT [17]
H	CIN [14]	CIN [19]	CIN [21]	ADR [2]	ADR [7]	$\overline{E2}$	GND	Vcc	Vcc	COUT [20]
J	CIN [16]	CIN [20]	$\overline{RESET}$	ADR [3]	ADR [6]	$\overline{E1}$	D[2]	D[5]	D[7]	COUT [21]
K	CIN [18]	ADR [0]	ADR [1]	ADR [4]	ADR [8]	$\overline{W}$	D[0]	D[1]	D[3]	D[4]

Note
All **V_{CC}** pins **must** be connected to the 5 Volt power supply.
All **GND** pins **must** be connected to ground.

A110-01.EPS

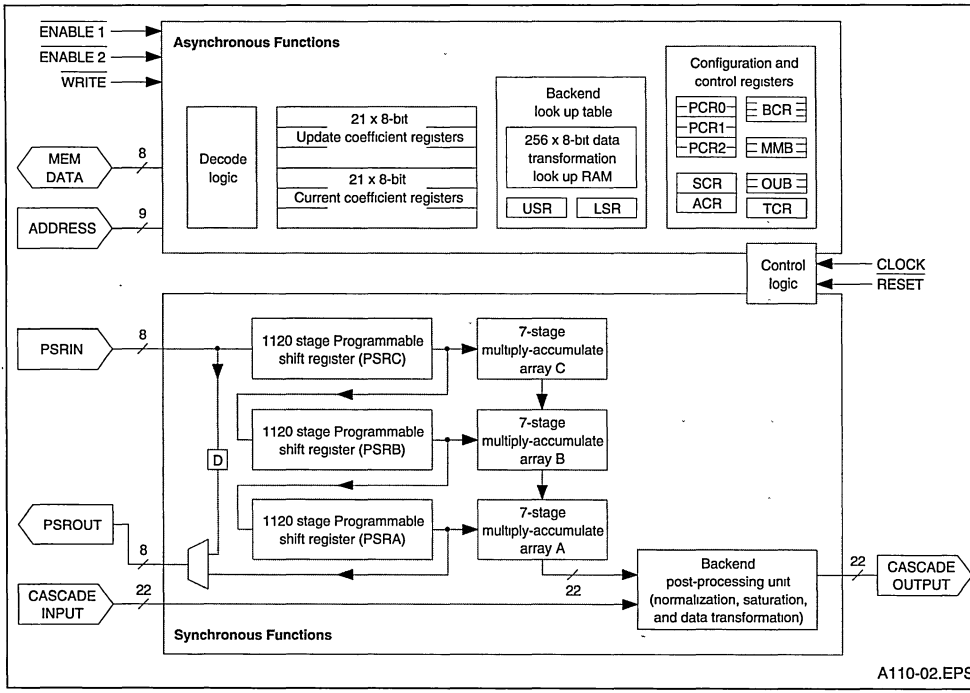
1. INTRODUCTION

The IMS A110 is a single-chip reconfigurable and cascadable subsystem suitable for many high speed image and signal processing applications. Apart from its powerful multiply-accumulate capability (420 MOPs), the strength of the IMS A110 lies in its extensive programmable support for data conditioning and transformation.

2. DESCRIPTION

The IMS A110 consists of a configurable array of multiply-accumulators, three programmable length 1120 stage shift registers, a versatile post-processing unit and a microprocessor interface for configuration and control purposes. The comprehensive on-chip facilities make a single device capable of dealing with many image processing operations.

Figure 1 : IMSA110 Users Model



The IMS A110 has five interfaces through which data can be transferred, Figure 1. The microprocessor interface allows access to the coefficient registers, the configuration and status registers, and the data transformation tables. The remaining four interfaces allow high speed data input and output to the IMS A110 and the cascading of several devices. A typical IMS A110 system is shown in Figure 3. If N devices are used in the cascade, they can be configured, entirely under software control, as a 21N stage 1-D transversal filter or as a 7X by 3Y 2-D window, where X and Y are any integers satisfying $N \leq XY$. For example 4 cascaded devices can be software configured as: an 84-stage 1-D filter, a 7 by 12 2-D window, a 28 by 3 2-D window, or a 14 by 6 2-D window.

The final output of the chip is 22 bits wide in twos complement format.

Figure 2 shows the distribution of the delays inside the part.

The latency between PSRin and COUT is dependent upon the length of PSRc. For example, with PSRc set to 0, and all coefficients set to zero except CR0c[6] (so the data passes through all MAC stages), the COUT bus will correspond to the PSRin bus delayed by 47 clock cycles.

The latency between PSRin and PSRout is 5 cycles PLUS the lengths of PSRc, PSRb and PSRa. If the shift registers are bypassed by setting SCR[1] to 1 then PSRout will be PSRin delayed by 2 clock cycles.

The Latency between the cascade input (CIN) and cascade output (COUT) is 6 cycles. This is shown lumped at the cascade input and cascade output pads in Figure 2. Figure 4 gives details of the data pipelining through the backend datapath.

Figure 2 : Synchronous Functions of the IMSA110

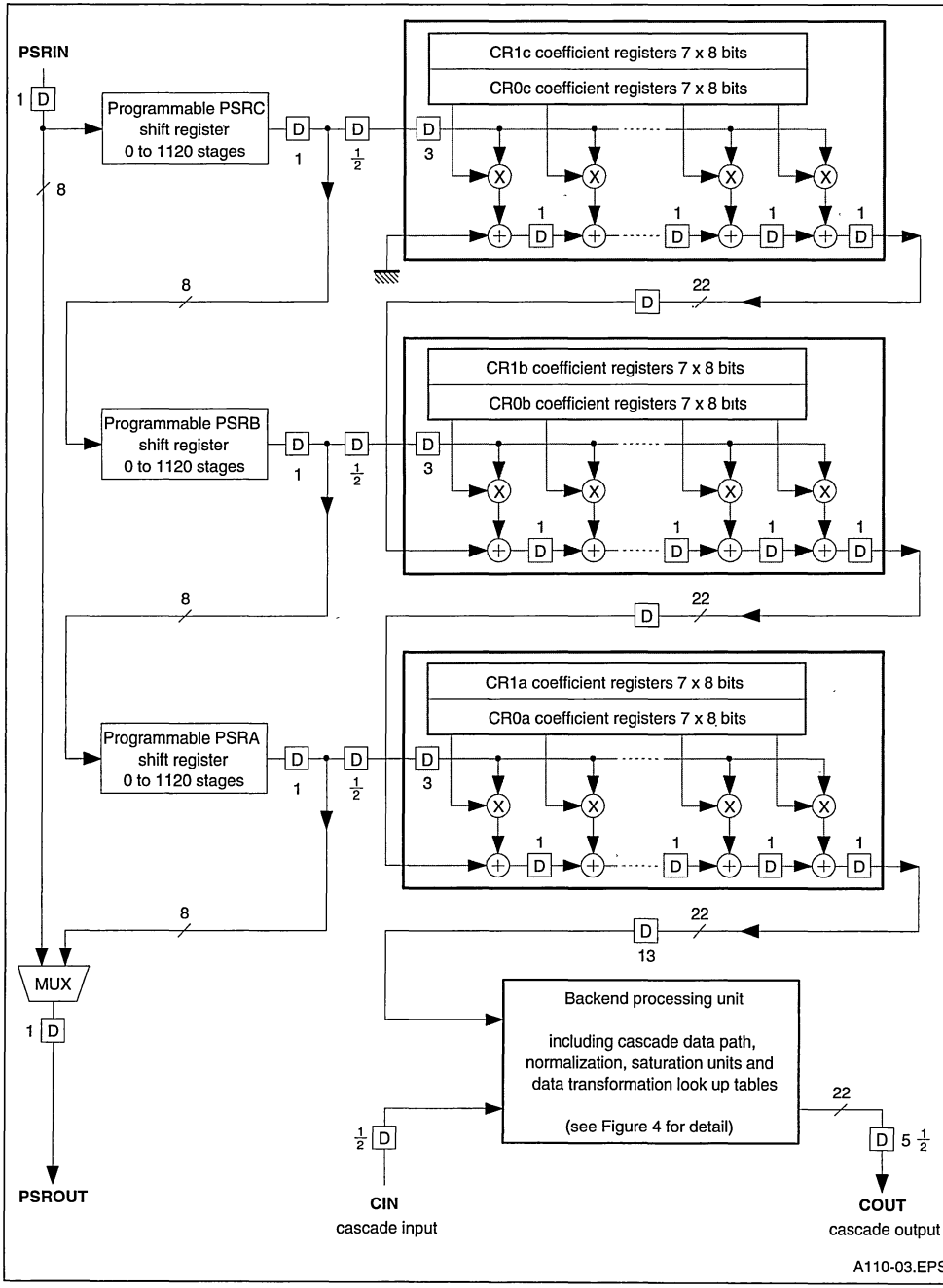
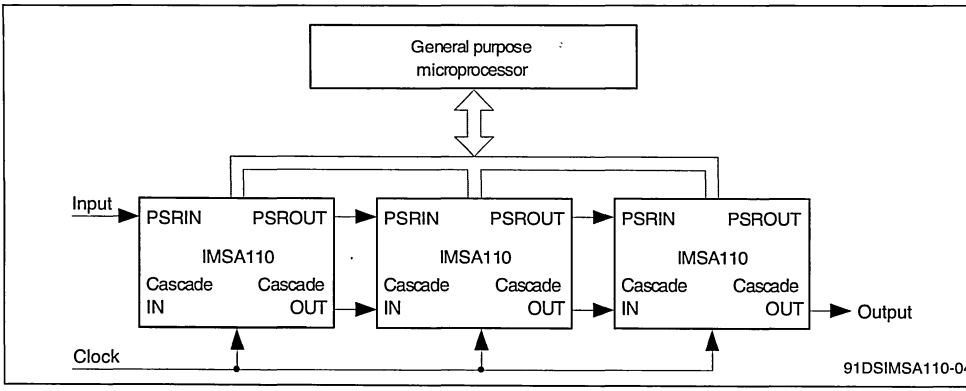


Figure 3 : A Typical IMSA110 Based System



3. PROGRAMMABLE SHIFT REGISTERS

The three shift registers are 8 bits wide and are each programmable from 0 up to 1120 clock cycles in length. The lengths are programmed into control registers via the microprocessor interface.

Data is clocked into the device via the PSRin bus (Programmable Shift Register in) at a maximum rate of 20 MHz. On-chip, the input data is then fed through a pipeline of the three shift registers. The output of the first shift register passes to the first 7-stage mac array and also to the input of the second shift register. Having passed through all three shift registers the data is output on the PSRout bus and can be used for cascading. Alternatively, as shown in Figure 2 the shift registers can be bypassed and the input data transferred to the PSRout bus after two delay stages. This mode can be controlled via the on-chip control registers and significantly simplifies software configuration of a cascade arrangement.

4. MAC ARRAY

As shown in Figure 2, the processing core of the device consists of a configurable array of multiply-accumulators (macs). The mac array consists of three 7-stage transversal filters which can be configured either as a 21-stage linear pipeline or as a 3 ×7 two-dimensional window. The input data is 8 bits wide and is fed to the mac array via three programmable shift registers.

The output of each shift register is supplied as input to one of the three 7-stage transversal filters. For each of the three transversal filters the associated input data is fed simultaneously to all 7 mac stages. At each stage the input sample is multiplied by a coefficient stored in memory, and added to the output of the previous stage delayed by one clock cycle. The output of each 7-stage mac is fed, via a delay stage, to the first stage in the next transversal filter.

The coefficient word width in the mac array is 8 bits wide. Two banks of coefficients are provided. At any instant one set of coefficients is in use within the mac array. The set in use is defined by the state of the 'Current Bank' bit, ACR[0]. The other set can be altered via the microprocessor interface. Once a new set of coefficients has been loaded, the activities of the two coefficient banks can be interchanged without interrupting the flow of data. Alternatively, by setting the 'continous bank swap' bit SCR[0], the two coefficient banks are swapped automatically after each data input. In this case the 'Current Bank' bit only determines which bank is used first. Both data input and coefficients can be programmed independently to support twos complement or positive unsigned formats allowing multiple devices to be used as a 'slice' in higher accuracy systems.

Within the mac array no truncation or rounding is performed on the partial products. The mac array output is fed to the backend post-processing unit which is responsible for data transformation / normalisation and cascading function.

5. BACKEND POST-PROCESSOR — hardware description

The Backend Post-Processor consists of four major blocks : The input block (shifter, cascade adder and rectifier unit), a statistics monitor, the data conditioning unit which itself consists of the data transformation unit and the data normaliser, and the output block (output adder and multiplexers).

A detailed diagram of the Backend Post-Processor is given in Figure 4.
All operations performed in the backend are on twos complement signed numbers unless otherwise stated.

5.1 Shifter, Cascade Adder and Rectifier

Data from the mac array enters the datapath via a programmable shifter. The shifter is capable of arithmetic right shifts (divides) of up to 8 bits with rounding, and left shifts of up to 8 bits. The size of this shift is controlled by the status bits BCR0[5-1]. The output of the shifter passes into the cascade adder where it is added, along with any rounding generated by the shifter, to either the cascade input bus (BCR0[0] = 0), or a zero value (BCR[0] = 1).
If the result of this 22-bit signed addition is greater than $2^{21} - 1$, (2097151_{10}) then the adder will generate a positive overflow. Likewise, if it is less than -2^{21}, (-2097152_{10}) a negative overflow will be generated. In other words, a positive overflow is generated if the result of adding two positive numbers (both MSBs = 0) is negative (resulting MSB = 1). Conversely, a negative overflow is generated if the result of adding two negative numbers (both MSBs = 1) is positive (MSB = 0). Adding two numbers of different signs cannot cause the adder to overflow.
The output of the cascade adder can optionally be full-wave or half wave rectified under the control of BCR0[7,6]. The output of the rectifier passes onto the X bus. Overflows on the X bus are signalled to both the statistics monitor and the data conditioner.

5.2 Statistics Monitor

The statistics monitor allows the user to set up watch dogs on the dynamics of the data on the X bus. It cannot affect the data on the X bus. The statistics gathered provide information on the system behaviour which can be used to ensure correct data scaling and normalisation. The information is also useful in the control of the overall system's analogue frontend.

Hardware/Functions

The statistics monitor consists of a 24 bit Min/Max register (MMR), a 24 bit Min/Max Buffer (MMB), a 22 bit Over/UnderShoot Counter (OUC), a 22 bit Over/UnderShoot Buffer (OUB) and a 22 bit twos complement comparator.
It can perform one of four functions :

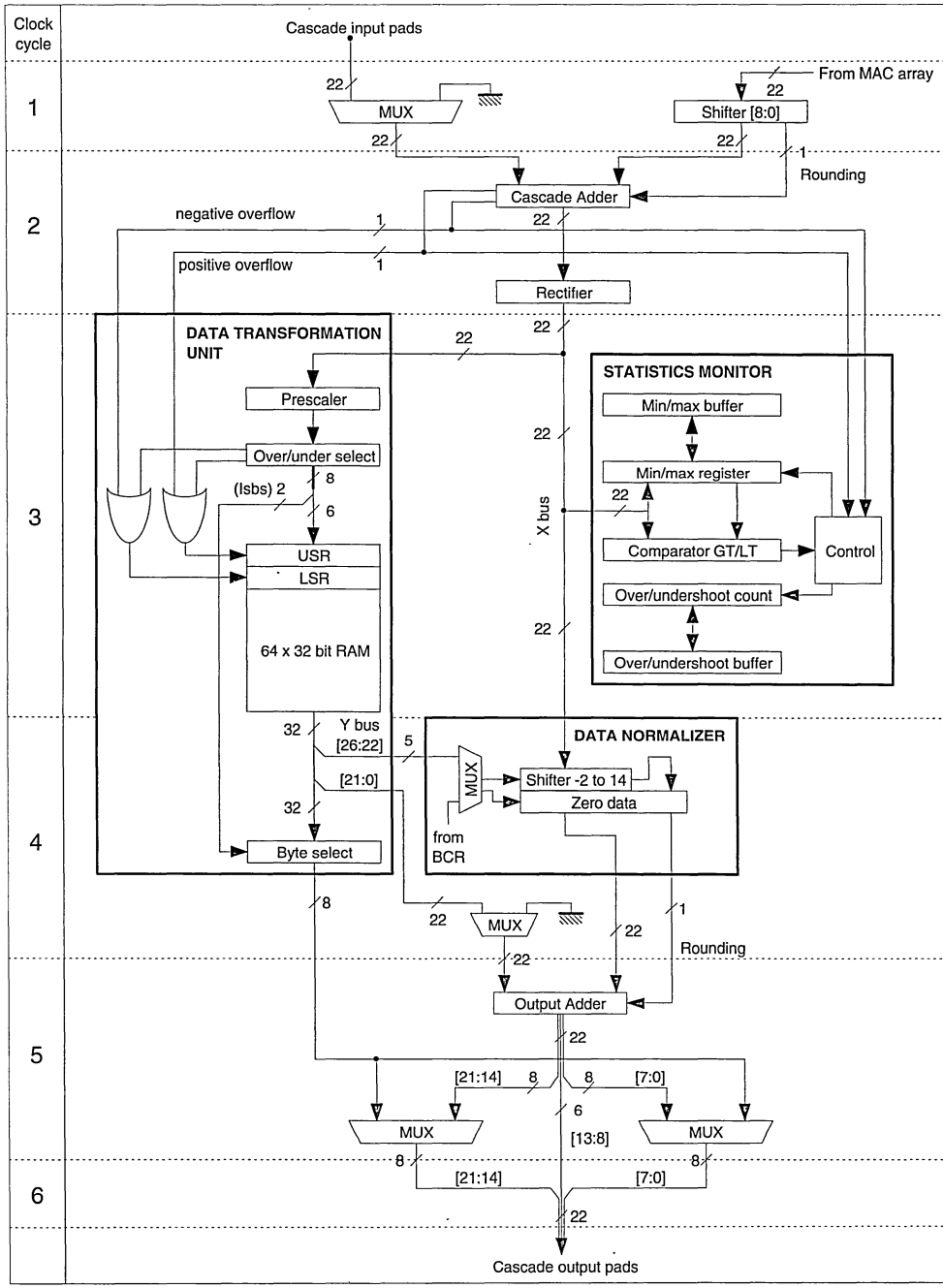
- **MAX REGISTER :** Capture the maximum value of data and store it in the MMR.
- **MIN REGISTER :** Capture the minimum value of data and store it in the MMR.
- **OVERSHOOT COUNTER :** Increment the OUC each time the data value exceeds the preset value in the MMR.
- **UNDERSHOOT COUNTER :** Increment the OUC each time the data value is less than the preset value in the MMR.

The mode of operation is determined by the Max/Min switch BCR1[0], and the Static Threshold switch BCR1[1].

Operation

Each sample on the X bus is compared against the threshold stored in the MMR.
If the unit is configured as an **overshoot counter** and the data on the X bus exceeds the threshold in the MMR, then the counter (OUC) is incremented. If the data is less than or equal to the threshold, then no action will occur. The OUC is unsigned and will not wrap around. Thus it behaves as a saturating counter with a maximum value of $2^{22} - 1$, ($3FFFFF_{16}$, 4194303_{10}). If there is a positive overflow on the X bus, then the counter will increment since the correct X bus value must exceed the threshold. Similarly a negative overflow on the X bus will not increment the counter since the correct X bus value cannot exceed the preset threshold.
If the unit is configured as an **undershoot counter** then the counter will be incremented whenever the sample is less than the preset threshold. In this case a negative overflow will cause the counter to increment.
If the unit is configured as a **max register** and the X bus exceeds the current threshold in the MMR, then the value on the Xbus is loaded into the MMR and becomes the new threshold and the counter is incremented. If the threshold is not exceeded then no action occurs. Thus the value in the MMR is the maximum value that has appeared on the X bus, and the value in the OUC has been incremented by the number of times that the threshold has been updated.
If the unit is configured as a **min register** then the threshold is updated and the counter incremented whenever the X bus is less than the current threshold.
When operating as a min/max register, overflows on the X bus can never cause the threshold to be updated as this would load an erroneous value into the MMR.

Figure 4 : Detailed Block Diagram of the Backend Post-processing Unit



Overflows

Bit 22 of the MMR records the history of positive overflows on the X bus. Similarly bit 23 records the history of negative overflows. These bits in the MMR are set to zero by writing to the MMR copy location and are active independently of whether the **Static Threshold** bit is set. When the MMR is read, then bits 22 and 23 are interpreted as follows:

bit 23	bit 22	condition
0	0	No overflow has occured
0	1	One or more positive overflows have occured
1	0	One or more negative overflows have occured
1	1	Both postive and negative overflows have occured

Detailed block diagram of the Backend Post-processing Unit

Access to registers

The MMR and OUC are accessed, through the memory interface, only via their associated buffers (MMB and OUB respectively) and are not accessible directly. In order to load the MMR with a value, the host must first write the value to the MMB and then transfer the data from the MMB to the MMR by performing a WRITE to the **copy MMR** location, $0B4_{16}$. To read the MMR the host must first perform a READ cycle from location $0B4_{16}$ (which transfers the contents of the MMR into the MMB) and then read the MMB. The OUB is accessed in the same way except that the dummy writes and reads are done to and from location $0BC_{16}$.

Copies from MMR to MMB and OUC to OUB (reads) can be performed at any time giving a snapshot of the contents of the MMR and OUC respectively. Copies from MMB to MMR and OUB to OUC (writes) can also be performed at any time allowing the threshold and counter to be updated dynamically.

5.3 Data transformation unit

The data transformation unit consists of a prescalar, an under/over select detector, a look up table and a byte selector. It can be used in isolation to perform abitrary data mappings, or in conjunction with the data normaliser to implement sophisticated dynamic range compression functions.

Prescalar

This allows an 8-bit field anywhere within the 22-bit X bus to be selected as the address to the LUT. This is performed by right shifting the X bus so that the required 8 bits are at the least significant end. The amount of right shift is programmed in BCR2[4-0] and can have a value from 0 to 16.

Over/under select detector

With PosLUTAddr (SCR[6]) set to zero, this unit monitors whether the amount of right shift performed by the prescalar is sufficient to include all significant bits in, and maintain the sign of, the selected 8 bit field (i.e. an over or under select is generated if the most significant bit of the selected 8 bit field differs from any subsequent bit right up to and including the most significant bit of the right shifted X bus). This will be an **overselect** if the X bus is positive (Bit 21 = 0), and an **underselect** if the X bus is negative (Bit 21 = 1). In other words the LUT address is always deemed to be signed with an address range of -128 to 127.

If however the control bit PosLUTAddr (SCR[6]) is set to one, the unit monitors whether the amount of right shift performed by the prescaler is sufficient to include all significant bits in the selected 8 bit field AND that all unselected bits are zero (i.e. an over or under select is generated if the first selected bit (bit 9) is not zero OR differs from any subsequent bit right up to and including the most significant bit of the right shifted X bus). This will be an **overselect** if the Xbus is positive and an **underselect** WHENEVER the Xbus is negative. Thus, in this mode, the address range of the LUT is 0 to 255.

Prescalar under/over selects and X bus positive/negative overflows are passed to the LUT along with the selected 8 bit address field.

Look up table (LUT) and byte select

The LUT consists of 64 words, 32 bits wide plus two special 32 bit locations called the **upper** and **lower saturation** registers (USR and LSR respectively). Thus the LUT is actually 66 words by 32 bits. The 32 bit output of the LUT is called the Y bus.

The most significant 6 bits of the 8 bit address field are used to address one of 64 words in the LUT. The least significant pair of bits in the 8 bit field are used to control a byte select on the output. Thus in addition to operating as a 64+2 word look up table of 32 bit words, it can be used as an 8 bit, 256+2 byte LUT providing 8bit — 8bit transformations.

Positive overflows on the X bus, and over selects in the prescalar cause the LUT to access the USR overriding the address given by the prescalar. Likewise negative overflows and under selects cause

the LUT to access the LSR. Any sort of overflow on the X bus or prescalar will cause the byte select control to be overridden and the **most significant byte** (byte 3) of the appropriate Saturation Register will appear on the byte wide output of the data transformation unit.
If there are simultaneous overflows on the X bus and in the prescalar then the overflow from the X bus takes priority.
The USR and LSR can thus be used to model the saturating behaviour of analogue circuits instead of the usual 'wrap around' encountered in digital systems. Alternatively the USR and LSR could signal error conditions within the backend directly on the output pins via one of the output multiplexers.
The LUT is loaded via the memory interface. The addressing for the LUT corresponds to the 8 bit field, assuming that the byte selector is being used. In order to access the look up table, USR and LSR from the microprocessor interface, the **LUT Access** control bit ACR[1] must be set to zero. This will force the Y bus to zero and the normaliser to be controlled by BCR3[7-3] regardless of the setting of the dynamic normalisation bit, BCR3[2]. The LUT, USR and LSR can then be loaded with any arbitrary value via the microprocessor interface. Setting the LUT access control bit to one will then allow the LUT to be used in the data transformation unit.

5.4 Data normaliser

This unit consists of a shifter capable of right shifts of up to 14 bits and left shifts up to 2 bits, followed by a *zero data unit* and an adder. The shifter is controllable from one of two 5 bit sources : control bits BCR3[7-3] or bits 26 to 22 of the Y bus. The control bit **Enable Dynamic Normalisation** (BCR3[2]) determines which source is in control of the normaliser. If this bit is set to zero the normaliser is controlled by BCR3[7-3]. The five bit field is a twos complement number between 14 and -2. This indicates the amount of right shift (negative meaning left shift). Any value outside this range causes the output of the shifter to be forced to zero. The output of the shifter, with any rounding generated by the shifter, goes into the output adder.

5.5 Output adder

This is a 22 bit adder with one of its inputs coming from the data normaliser. The other input is either bits 21 to 0 of the Y bus from the data transformation unit, or set to zero under the control of BCR3[1]. Note that any overflow occuring due to left shifting in the normaliser or the subsequent addition in the output adder is not detected by the IMS A110.

5.6 Output multiplexers

These two multiplexers allow the currently selected byte from the LUT to be optionally selected to drive either the most significant byte and/or the least significant byte of the Cascade Output pins. This is controlled by the state of BCR2[5] and BCR2[6]. Enabling either of these multiplexers overrides the state of the Cascade Output pins only on the relavent 8 pins. The remaining pins will continue to represent the output of the output adder.

6. BACKEND POST-PROCESSOR — Modes of Operation

The backend post-processing unit is capable of performing many functions including data scaling, transformation, dynamic range compression and histogram equalisation.

6.1 Default mode (after Reset)

At power up or after reset the state of the backend post-processor is such that data from the MAC array and the cascade input are added and pass straight through the datapath unaffected.
The default mode for the statistics monitor is **min register** although the values in the OUB, OUC, MMR and MMB will be undefined. Likewise the contents of the LUT, USR and LSR will be undefined, the **LUT Access** control bit will be zero forcing the Y bus to zero and allowing the microprocessor interface to access the LUT, USR and LSR.
Note that the cascade output pins and the PSR output pins are tristated.

6.2 Cascade adder / MAC data scalar

These units allow the cascading of IMS A110s where the output of the MAC array may be scaled before it is added to the cascade input data. The shifter can also be used for combining devices to obtain extended precision in input data, coefficient word length or both.
The ability to zero the cascade input provides a simple means of controlling the number of 'active' devices cascaded as well as a means of debugging large systems.

6.3 Rectification

Rectification, the removal of negative results, is needed in several image processing functions.
For example, edge detection using a Sobel oper-

ator usually requires full wave rectification due to the different signs obtained at differing edge transitions. Edge detection using a Laplacian operator produces a change of sign at an edge. In this case, removing negative numbers using half wave rectification can produce better results as full wave rectification can lead to some blurring of the edge transition.

6.4 Static scaling

This can be performed using one of two units: the MAC array output shifter (as above), and the data normaliser. In the second case the data undergoes a simple scaling operation (with rounding) within the normaliser. The normaliser can be used to scale (multiply) the data by the factors 0, 1/16384, 1/8192, 1/4096 ..., 1/2, 1, 2, 4. By controlling the normaliser from the control bits BCR3[7-3], this provides a means for simple scaling of the data before it is output. Setting BCR3[1] and BCR2[6,7] to zero ensures that the data transformation unit takes no part in the operation and the output of the normaliser is passed unchanged to the output pins.

6.5 Dynamic scaling

In this mode the scaling is controlled by the data itself. i.e. the scalar is controlled from the LUT (Ybus bits 26-22) by setting BCR3[2] to one, the Ybus input to the output adder being set to zero either by setting BCR3[1] to zero or programming the LUT accordingly. This mode can provide a discontinuous non-linear transformation.

6.6 Simple transformation

This mode allows the user to apply arbitrary transformations to the data before it is output. Here the LUT is treated as 256 by 8, addressed as either -128 to 127 if PosLUTAddr is set to zero or 0 to 255 if PosLUTAddr is set to one. The 8 bit field selected by the LUT prescalar is used to address a byte in the LUT which is passed directly to the output pins via one of the output multiplexers. Ybus control of the data normaliser is disabled, BCR3[7-3] are set out of range so as to zero the normaliser output and the Ybus input to the output adder is set to zero by BCR3[1]. One (or both) of the output multiplexers are enabled and so the addressed byte from the LUT passes straight to the cascade output pads. Only the most significant byte of the USR and LSR are applicable in this mode as overflows override the byte select control and force it to select the most significant byte.

6.7 Dynamic normalisation

In this mode the normaliser and transformation units in the output conditioner are used together to perform sophisticated non-linear dynamic range compression and transformations. As in the simple transformation case the prescalar selects an 8 bit field anywhere within the X bus. The most significant 6 bits, and overflows, are fed as an address to the LUT. In this case the look up table is treated as 64+2 by 32. Bits 26 to 22 of the Y bus are used to control the normaliser block so that the input to the normaliser is dynamically scaled. The output of the normaliser is then added in the output adder to the least significant 22 bits of the Y bus (Note that only 28 bits of the 32 bit Y bus are actually used).

Thus the data is scaled, rounded, and then an offset is added to the scaled result. Each operation can be viewed as

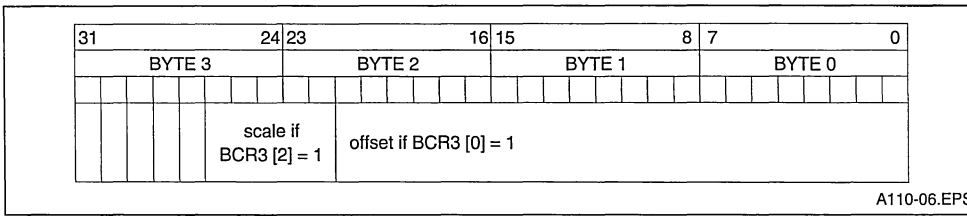
$$\text{output} = \text{input} \times \text{scale} + \text{offset}$$

Where scale and offset are both programmable functions of input. One way to view this operation is to consider that the original data range is divided into 64 equal sized levels and in each level a different scale and offset is applied. The scale and offset stored in the USR and LSR would be chosen to give the desired behaviour under overflow conditions.

Note that in the case of cascade adder overflows, the data on the X bus is invalid, so the scale here would usually be set out of range so as to zero the normaliser output. The offsets in the USR and LSR would then provide the cascade output directly.

Note also that if the 5 bit scale field in the LUT is programmed so that the normaliser always zeros the data, then the output will correspond to the 22 bit offset field in the LUT. This can be viewed as a coarse transformation with wide dynamic range which is useful for applications such as image contour emphasis and equalisation.

Figure 5 : Bit Format of Data Stored in LUT, USR and LSR



7. GLOSSARY

This section defines the meaning of terms used elsewhere in this data sheet.

Arithmetic Shift

For a right shift, the most significant bit is always copied into the most significant end of the result. For example shifting right by 2:

01000101	→	00010001
11000101	→	11110001

For a left shift, the least significant bit will become zero.

Note that left shifting can cause overflows and these are not detected in the MAC output scalar or the data normaliser.

Rounding

All rounding done within the IMS A110 is equivalent to truncating after adding 1/2 LSB. (Rounding is always applied in the positive direction). For example for 8 bit twos complement numbers undergoing a two bit right shift:

00000011 → 00000000 + 1 = 00000001 (rounded up)
00000010 → 00000000 + 1 = 00000001 (rounded up)
11111110 → 11111111 + 1 = 00000000 (rounded up)
00000001 → 00000000 (no rounding)
11111101 → 11111111 (no rounding)

Left shifts do not generate rounding.

Transversal Filter

A transversal filter is a calculation consisting of the sum of products of successive points of input data. For input data x_i, x_{i+1}, ..., and a set of coefficients, c_6, c_5, ..., the result, Y is:

$$Y = \sum_{i=0}^{6} c_i \times x_{6-i}$$

Two's Complement

Two's complement numbers allow both positive and negative numbers. For example in 8 bit numbers the most positive number is 127, the most negative is -128:

two's complement	decimal
10000000	-128
10000001	-127
11111111	-1
00000000	0
00000001	1
01111111	127

Rectification

Rectification is a method of removing negative numbers. There are two methods: Full wave and Half wave. In either case all positive numbers and zero are unaffected. In Full wave rectification, any negative numbers are negated (i.e. multiplied by 1) so that they become positive. In Half wave rectification, all negative numbers are replaced by zero.

Dynamic Range Compression

When Dynamic is used in this context, it is to indicate a change of behaviour for each data point. For example, a dynamic shift is one where the size of the shift may change on each successive clock cycle. Dynamic range compression is range compression making use of an offset and shift, which can change depending on each data point. This allows the essential non-linear transformations required in image processing to be implemented on the IMS A110.

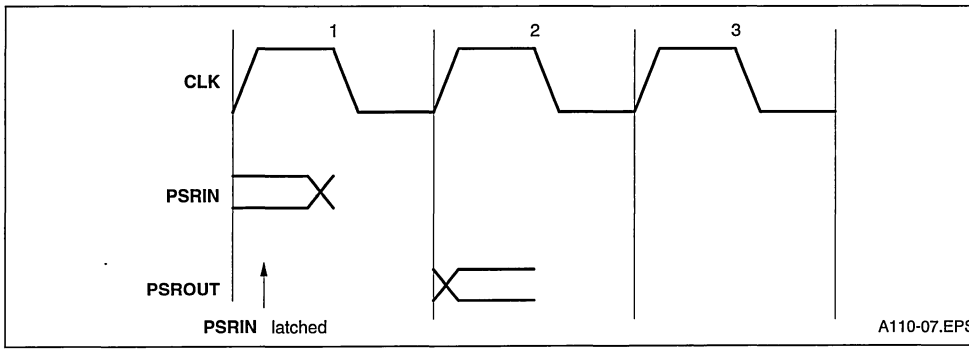
Bit Fields

Bits, words and addresses in this data sheet are little-endian; The lowest order byte of a multiple byte word is referred to as byte 0, and is addressed in the same way. Similarly, the least significant bit of any bit field is that with the lowest bit number. For example, 'bits 26-22' refers to a 5 bit field where bit 22 is treated as the least significant, and bit 26 as the most significant.

Latency

Within the IMS A110 the latency is the number of clock cycles from an input to its corresponding output. For instance, with the programmable shift registers bypassed by setting SCR[1] to 1, the latency from PSRin to PSRout will be 2 as shown in Figure 6.

Figure 6



PIN DESIGNATIONS

System services

Pin	In/out	Function
V_{CC}, GND		Power supply and return
CLK	in	Input clock
$\overline{\text{RESET}}$	in	System reset

Synchronous input/output

Pin	In/out	Function
PSRin[7-0]	in	Programmable shift register input
PSRout[7-0]	out	Programmable shift register output
Cin[21-0]	in	Cascade input port
Cout[21-0]	out	Cascade output port

Asynchronous input/output

Pin	In/out	Function
$\overline{\text{E1}}$, $\overline{\text{E2}}$	in	Memory interface enable signals
$\overline{\text{W}}$	in	Memory interface write enable
ADR[8-0]	in	Memory interface adress bus
D[7-0]	in/out	Memory interface data bus

Notes
Signal names are shown with an overbar if they are active low, otherwise they are active high.

8.1 System services

System services include all the necessary logic to start up and maintain the IMS A110.

Power

Power is supplied to the device via the **V_{CC}** and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100nF low inductance (e.g. ceramic) capacitor between **V_{CC}** and **GND**.

CLK

The clock signal **CLK** controls the timing of input and the output on the four dedicated interfaces, and controls the progress of data through the shift registers, multiply-accumulate array and post-processing unit. The A110 is fully static so the clock can be slowed down or stopped in either state without corrupting data.

$\overline{\text{RESET}}$

If this pin is taken low for at least 2 clock cycles, the control logic within the IMS A110 will be reset and all of the control and configuration registers will be initialised to their default values. All other register, memory locations, datapath registers and shift registers will not be reset by this signal.
A reset is initiated automatically when power is first applied to the device. This reset will be completed once four cycles of **CLK** have occured after **VCC** is valid.

8.2 Synchronous services

PSRin[7-0]

This 8-bit wide bus supplies input data to the device. The input data enters the first of the three shift registers in the chain. The timing of this input is controlled by the **CLK** signal. The data on the **PSRin** port is sampled on the rising edge of the clock. In a cascade arrangement, this bus will be connected to the **PSRout** port of the previous device. In such an arrangement the **PSRin** port on the first device will be the input to the overall cascaded system.

PSRout[7-0]

This bus outputs the data from the last programmable shift register in the chain. The data on this bus is synchronously clocked by the rising edge of **CLK**. In a cascade arrangement this port will be connected to the **PSRin** port of the next device. At power up, or after a reset, the **PSRout** pins are tristated. They are enabled by SCR[5].

Cin[21-0]

The Cascade Input port allows IMS A110s to be cascaded. It also can be used for combining an external signal (e.g. a reference image or an offset) with the processed result. In a cascade arrangement, this bus will be connected to the Cascade Output of the previous device. The data on the **Cin** bus is sampled on the rising edge of **CLK**.

Cout[21-0]

This bus outputs the processed result from the IMS A110 and can also be used for cascading. The 22-bit result is synchronously clocked by the rising edge of **CLK**. In a typical cascaded system this bus will be connected to the Cascade Input port of the next device. On the last device in the cascade, this bus will be the output of the overall system. At power up, or after a reset, the **Cout** pins are tristated. They are enabled by SCR[4].

8.3 Asynchronous input/output

$\overline{E1}$, $\overline{E2}$

If both of these signals are low, then the microprocessor interface is enabled. The operation of these enable signals is very similar to those found on static RAMs. When either of these signals are high the Write Enable and the address inputs are ignored and the microprocessor interface Data signals are high impedance. When both Enable signals are low a read or write access is made to registers or the RAMs within the IMS A110. Access to the microprocessor interface can occur asynchronously to the synchronous pins (**PSRin**, **PSRout**, **Cin**, **Cout**) of the device.

$\overline{W}$

Write Enable indicates whether the access to the IMS A110 memory interface is to be a read or a write. If $\overline{W}$ is low a write access is indicated.

ADR[8-0]

The nine bit binary value applied to the address inputs of the IMS A110 indicates which register or RAM location within the device is to be accessed.

D[7-0]

During a write to the microprocessor interface an 8-bit word is applied to the Data pins which is written to the appropriate location. During a read cycle the contents of the location accessed are placed on the Data pins. When either of the Enables are high the Data pins are high impedance.

9 REGISTER DESCRIPTION

Memory map

Within the IMS A110 addresses are fully decoded. Reading from locations not defined in the memory map will produce zero data. Data written to such locations is ignored. This allows the part to be fully programmed using a ROM with an address incremeter. In this case, for future compatibility, zero should be written to all undefined locations.

Register	Address decimal	Address hex	Function
CR0a	0—6	000—006	Coefficient Registers Bank 0a
CR0b	16—22	010—016	Coefficient Registers Bank 0b
CR0c	32—38	020—026	Coefficient Registers Bank 0c
CR1a	64—70	040—046	Coefficient Registers Bank 1a
CR1b	80—86	050—056	Coefficient Registers Bank 1b
CR1c	96—102	060—066	Coefficient Registers Bank 1c
PCRA	128—129	080—081	PSRA Control Register
PCRB	130—131	082—083	PSRB Control Register
PCRC	132—133	084—085	PSRC Control Register
SCR	144	090	Static Control Register
ACR	146	092	Active Control Register
BCR	160—163	0A0—0A3	Backend Configuration Register
MMB	176—178	0B0—0B2	Maximum/Minimum Buffer
CMM	180	0B4	Copy MMR
OUB	184—186	0B8—0BA	Overshoot/Undershoot Buffer
COU	188	0BC	Copy OUC
TCR	208	0D0	Test Control Register
USR	248—251	0F8—0FB	Upper Saturation Register
LSR	252—255	0FC—0FF	Lower Saturation Register
LUT	256-511	100—1FF	Look up Table

9.2 Registers

CR0a Coefficient registers bank 0a

These seven 8-bit locations contain coefficients which can be used by the third, of the three, 7-stage mac arrays. CR0a(0) (address #000) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR0a(6) (address #006) corresponds to the coefficient register of this mac nearest to its input. These Coefficient registers can be written to provided that the other register bank is in use. Whether the coefficient written is signed or unsigned is determined by the 'Unsigned Coefficient' bit SCR[3]. Once a value is written to a coefficient register, its value can be read back from an internal duplicate register. These registers will be used by the mac array, when ACR[0], 'Current Bank' is set to zero. Writing to these Coefficient Registers while in use will result in an undefined operation of the mac array.

CR0b Coefficient registers bank 0b

These seven 8-bit locations contain coefficients which can be used by the second, of the three, 7-stage mac arrays in the chain. CR0b(0) (address #010) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR0b(6) (address #016) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR0a.

Figure 7 : IMSA110 Memory Map

Address (Hex)	Name	bit 7	6	5	4	3	2	1	0
1FF 100	**LUT**	Look Up Table							
0FC-0FF	**LSR**	Lower Saturation Register							
0F8-0FB	**USR**	Upper Saturation Register							
0D0	**TCR**								
0BC	**COU**	Copy Over/UnderShoot Buffer							
0B8-0BA	**OUB**	Over/UnderShoot Buffer							
0B4	**CMM**	Copy Min/Max Buffer							
0B0-0B2	**MMB**	Min/Max Buffer							
0A3	**BCR3**	Normaliser Control					Dynamic normali-sation	LUT to output adder	0
0A2	**BCR2**	0	LS output byte	MS output byte	Look Up Prescaler				
0A1	**BCR1**	0	0	0	0	0	0	Static threshold	Greater Than
0A0	**BCR0**	Full Wave	Half Wave	MAC Output Scaler					Zero Cascade
092	**ACR**	0	0	0	0	0	0	Backend LUT Access	Current Bank
090	**SCR**	0	PosLUT Addr	PSR Out Enable	Cascade Enable	Unsigned Coef Load	Unsigned Data	Bypass PSRs	Cont Swap
085	**PCRC**	0	0	0	0	0	Shift Length (Upper Bits)		
084	**PCRC**	Shift Length (Lower Bits)							
083	**PCRB**	0	0	0	0	0	Shift Length (Upper Bits)		
082	**PCRB**	Shift Length (Lower Bits)							
081	**PCRA**	0	0	0	0	0	Shift Length (Upper Bits)		
080	**PCRA**	Shift Length (Lower Bits)							
066 ... 060	**CR1c**	Bank 1 Coefficient Register							
056 ... 050	**CR1b**	Bank 1 Coefficient Register							
046 ... 040	**CR1a**	Bank 1 Coefficient Register							
026 ... 020	**CR0c**	Bank 0 Coefficient Register							
016 ... 010	**CR0b**	Bank 0 Coefficient Register							
006 ... 000	**CR0a**	Bank 0 Coefficient Register							

CR0c Coefficient registers bank 0c

These seven 8-bit locations contain coefficients which can be used by the first, of the three, 7-stage mac arrays in the chain. CR0c(0) (address #020) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR0c(6) (address #026) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR0a.

CR1a Coefficient registers bank 1a

These seven 8-bit locations contain coefficients which can be used by the third, of the three, 7-stage mac arrays in the chain. CR1a(0) (address #040) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR1a(6) (address #046) corresponds to to the coefficient register of this mac nearest to its input. These registers will be used provided that ACR[0], 'Current Bank' is set to one, or continuous bank swap mode is in operation (SCR[0] set to one).

CR1b Coefficient registers bank 1b

These seven 8-bit locations contain coefficients which can be used by the second, of the three, 7-stage mac arrays in the chain. CR1b(0) (address #050) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR1b(6) (address #056) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR1a.

CR1c Coefficient registers bank 1c

These seven 8-bit locations contain coefficients which can be used by the second, of the three, 7-stage mac arrays in the chain. CR1c(0) (address #060) corresponds to the coefficient register of this mac array nearest to its output. Similarly CR1c(6) (address #066) corresponds to to the coefficient register of this mac nearest to its input. Their behaviour is otherwise identical to CR1a.

PCRA PSRA Control register

This is a 16-bit register, with least significant byte at location #080, and is used to set up the length of the last shift register in the chain. Programmed lengths outside the range 0 to 1120 will cause undefined behaviour of the shift register.

PCRB PSRB Control register

This is a 16-bit register, with least significant byte at location #082, and is used to set up the length of the second shift register in the chain. Programmed lengths outside the range 0 to 1120 will cause undefined behaviour of the shift register.

PCRC PSRC Control register

This is a 16-bit register, with least significant byte at location #084, and is used to set up the length of the first shift register in the chain. Programmed lengths outside the range 0 to 1120 will cause undefined behaviour of the shift register.

SCR Static control register

The Static Control Register contains the control bits which set up parts of the IMS A110 which are likely to not need reconfiguration during processing. The contents of this register are not affected by the IMS A110 and can be read at any time. Modifying the Static Control register during processing will result in undefined behaviour. Normal operation will start to occur between 0 and 3 clock cycles after the completion of the write cycle.

ACR Active control register

The Active Control Register contains status and control bits which are likely to be accessed during normal operation of the IMS A110.

BCR Backend configuration register

The Backend Configuration Registers consist of four byte-wide registers BCR0, BCR1, BCR2, and BCR3 which are located at addresses #0A0, #0A1, #0A2, and #0A3 respectively. These four registers are used to control the backend post-processing unit. None of the control bits in these registers can be modified by the IMS A110. Modification of the values in these registers during processing may result in undefined behaviour. Normal operation will start to occur between 0 and 3 clock cycles after the completion of the write cycle.

MMB Maximum/minimum buffer

These three locations hold a 24-bit wide word, with the least significant byte at the lowest address, and act as a buffer between the MMR and the microprocessor interface. All the transactions between the MMR and the host processor must take place through this register. When the MMR is not in use, the value of this buffer is undefined.

CMM Copy MMR

This location is used to enable the data transfer

between the MMB and MMR. A write to this location causes the contents of MMB to be copied into the MMR and bits 23 and 22 of the MMR (the cascade adder overflow flags) to be set to zero. A read from this location causes the reverse, i.e the contents of the MMR are copied into the MMB. The value written to this location is ignored, the value read back is undefined.

OUB Overshoot/undershoot buffer

These three memory locations hold a 22-bit word, with the least significant byte at the lowest address, and act as a buffer between the OUC and the microprocessor interface. All the transactions between the OUC and the host processor must take place through this register. When the OUC is not in use, the value of this buffer is undefined.

COU Copy OUC

This location in the memory is used to enable the data transfer between the OUB and OUC. A write to this location causes the contents of OUB to be copied into the OUC. A read from this location causes the reverse, i.e the contents of the OUC are copied into the OUB. The value written to this location is ignored, the value read back will be undefined.

TCR Test control register

This register is used for testing, and should be loaded with zero for normal operation.

USR Upper saturation register

This is a 32-bit value with the least significant byte at the lowest address. Its contents are used to replace the LUT output if positive overflow(s) occur in the look up prescaler and / or in the cascade adder. Accesses from the microprocessor interface can only be made while ACR[1] is set to zero.

LSR Lower saturation register

This is a 32-bit value with the least significant byte at the lowest address. Its contents are used to replace the LUT output if negative overflow(s) occur in the look up prescaler and / or in the cascade adder. Accesses from the microprocessor interface can only be made while ACR[1] is set to zero.

LUT Look up table

These locations are for the 256-byte look up table which is used for data mapping and transformation operations. From the microprocessor interface, these locations are addressed in the same way as that seen by the 8-bit output of look up prescaler. When used in 32 bit mode, the locations are treated in the same way as other 32 registers: Word 0 has its most significant byte at #103, its least significant byte at #100, Word 12 has its most significant byte at #133, its least significant byte at #130. Accesses from the microprocessor interface can only be made while ACR[1] is set to zero.

10. REGISTERS — BIT ALLOCATION

This section describes the register details bit by bit. Each section commences with the name of the register with the bit number(s) followed by the default value, in the general format:

Name REGISTER [MSB—LSB] Default : MSB...LSB

The least significant bit of a register is bit 0.

* in the tables indicates the default state of the register bit(s).

10.1 PSR control registers (PCR)

PSRA control PCRA[10-0] Default: 0...0

These eleven least significant bits of the PCRA are used to specify the length of the last Programmable Shift Register (PSRA). The length of the shift register will be numerically equal to the binary value loaded in these bits. The value loaded in must be in the range of 0 to 1120 decimal. If a value outside this range is written to these bits the behaviour of the shift register will be undefined. After updating this register, the behaviour of the delay is undefined for 22 clock cycles. Hence changing the length from 1000 to 1001 delays, will result in correct output only after 1023 cycles. This will also have to propagate through the backend before the cascade output values will be correct.

Reserved PCRA[15-11] Default: 00000

These 5 most significant bits of the PCRA are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

PSRB control **PCRB[10-0]** **Default: 0...0**

These eleven least significant bits of the PCRB are used to specify the length of the second Programmable Shift Register (PSRB). The length of the shift register will be numerically equal to the binary value loaded in these bits. The value loaded in must be in the range of 0 to 1120 decimal. If a value outside this range is written to these bits the behaviour of the shift register will be undefined. After updating this register will also have to propagate through PSRA and the backend before the cascade output values will be correct

Reserved **PCRB[15-11]** **Default: 00000**

These 5 most significant bits of the PCRB are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

PSRC control **PCRC[10-0]** **Default: 0...0**

These eleven least significant bits of the PCRC are used to specify the length of the first Programmable Shift Register (PSRC). The length of the shift register will be numerically equal to the binary value loaded in these bits. The value loaded in must be in the range of 0 to 1120 decimal. If a value outside this range is written to these bits the behaviour of the shift register will be undefined. After updating this register will also have to propagate through PSRB, PSRA and the backend before the cascade output values will be correct

Reserved **PCRC[15-11]** **Default: 00000**

These 5 most significant bits of the PCRC are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

10.2 Static control register (SCR)

Reserved **SCR[7]** **Default: 0**

This location is reserved. The user should write zero to this location to maintain compatibility with future products. The value read from this location will be zero.

Positive Look up table address **SCR[6]** **Default: 0**

This bit affects the way in which the over/under select detector checks the LUT address. It determines whether the address range of the LUT is signed (-128 to 127) or positive (0 to 255). A one at this location indicates a positive LUT address.

PSR out Enable **SCR[5]** **Default: 0**

A zero at this location will force the PSR Output pins into the tristate mode.

Cascade Enable **SCR[4]** **Default: 0**

A zero at this location will force the Cascade Output pins into the tristate mode.

Unsigned coefficient load **SCR[3]** **Default: 0**

If this bit is set to one, the format of subsequently loaded coefficients become unsigned, with coefficient value assuming a range between 0 and 255 decimal. An 8-bit coefficient with all its bits set to one will represent +255 decimal. When this bit is zero the format of subsequently loaded coefficients will be twos complement and the corresponding numerical value will have a range between -128 and +127. By changing this bit whilst coefficients are being loaded, coefficients between -128 and +255 can be used. The unsigned format on all coefficients is suitable when IMS A110s are combined to obtain wider coefficients for extended precision.

SCR[3]	Coefficient type
0	Signed coefficients *
1	Unsigned coefficients

Unsigned data **SCR[2]** **Default: 0**

If this bit is set to one, the IMS A110 input data format will become unsigned, with input data value assuming a range between 0 and 255 decimal. An 8-bit value with all its bits set to one will represent +255 decimal. When this bit is zero the input data format will be twos complement and the corresponding numerical value will have a range between -128 and +127. Unlike SCR[3], this bit cannot be used to dynamically alter the data format. The unsigned format is suitable when IMS A110s are combined to obtain wider input data for extended precision.

SCR[2]	Data type
0	Signed data *
1	Unsigned data

Bypass shift registers **SCR[1]** **Default: 0**

This bit is used to program the path between the PSRin and PSRout ports. A zero at this location will cause the output from the last programmable shift

register to be sent to PSRout port. Writing a one to this bit will cause the three programmable shift registers to be bypassed, and the data entering the port PSRin to be fed directly, via a delay of 2 clock cycles, to the port PSRout. This bit allows full programmability of a cascade arrangement so that the same hardware can be operated in a variety of ways.

Continous bank swap SCR[0] Default: 0

The continuous bank Swap bit selects whether the the two banks of coefficient registers are used alternately after each data input or if this is controlled solely by the state of the 'Current Bank' bit in the Active Control Register ACR[0].

SCR[0]	Swap mode
0	Swap on asserting ACR[0] *
1	Swap after end of each input cycle

10.3 Active control register (ACR)

Reserved ACR[7-2] Default: 00000

These 6 most significant bits of the ACR are reserved. The user should write zero to these locations to maintain compatibility with future products. The value read from these locations will be zero.

Enable look up table ACR[1] Default: 0

Writing a zero into this control bit allows the memory interface to access the Look up table; the output to the data transformation unit will be zero. The normaliser will be controlled by BCR3[7-3], regardless of the state of BCR3[2]. Writing a one to ACR[1] allows the IMS A110 to use the Look up Table. After changing this bit, 2 clock cycles must occur before the Look up Table can be accessed.

ACR[1]	LUT mode
0	Memory interface access *
1	Data transformation unit

Current bank ACR[0] Default: 0

When the 'Continuous Bank Swap' bit is set to zero, writing a zero into this control bit instructs the IMS A110 to use the set of coefficient registers at addresses 0 to #X26. Setting a one to this bit instructs the IMS A110 to use the set of coefficient registers at addresses #40 to #X66. If the 'Continuous Bank Swap' bit is set to one, then this bit only indicates the bank selected for the first cycle of the continuous swap mode. Writing to this bit whilst in continuous bank swap mode (SCR[0]=1) will result in undefined behaviour of the mac array.

ACR[0]	Coefficient bank
0	Use coefficient registers at 0 to #X26 *
1	Use coefficient registers at #40 to #X66

10.4 Backend control register 0 (BCR0)

Enable full-wave rectification BCR0[7] Default: 0

If this bit is set the output of the cascade adder is full-wave rectified (absolute value operation) before it is fed to the remainder of the backend. This bit will override the function of the BCR0[6].

Enable half-wave rectification BCR0[6] Default: 0

Writing a one in this bit will cause the negative values from the cascade adder to be replaced with zero. Note that writing a one into BCR0[7] will override the function of this control bit.

BCD0[7-6]	Rectifier mode
0 0	Straight through *
0 1	Half wave rectification
1 0	Full wave rectification
1 1	Full wave rectification

Mac array output scaler BCR0[5-1] Default: 00000

The contents of these five bits control the amount of right or left shift applied to the data at the output of the mac array. This field is interpreted as a two's complement number. A positive number represents a right shift (divide). Any shift in the range -8 (11000) to +8 (01000) is legal. Values outside this range will result in undefined behaviour of the mac output scaler.

Zero cascade input BCR0[0] Default: 0

This bit controls the Cascade Input Multiplexer. Writing a one to this bit will cause a zero, instead of the cascade input data, to be fed to the cascade adder.

BCR[0]	Cascade input mode
0	Cascade data *
1	Zero

10.5 Backend control register 1 (BCR1)

Reserved **BCR1[7-2]** **Default: 00000**

These locations are reserved. The user should write zero to these locations to maintain compatibility with future products. The values read from these locations will be zero.

Static threshold **BCR1[1]** **Default: 0**

If this bit is set to one, the signals from the comparator will be used to increment the Over / Undershoot Counter only. If this bit is zero, the signals from the comparator will be used to latch the output of the Cascade Adder into the Maximum / Minimum Register (MMR), **and** to increment the counter. In this case the counter will have been incremented by the number of times that the threshold has been updated.

Enable greater than **BCR1[0]** **Default: 0**

This control bit determines whether the comparator in the statistics monitor behaves as a 'greater than', or as a 'less than' comparator. The signal from this comparator is used to drive the Over / Undershoot Counter and the Max / Min Register. A one at this location selects 'greater than'.

BCR1[1-0]	Statistics monitor mode
0 0	Min. register *
0 1	Max. register
1 0	Undershoot counter
1 1	Overshoot counter

10.6 Backend control register 2 (BCR2)

Reserved **BCR2[7]** **Default: 0**

This location is reserved. The user should write zero to this location to maintain compatibility with future products. The value read from this location will be zero.

Pass LUT data to least significant output **BCR2[6]** **Default: 0**

This bit controls the output multiplexer. If this bit is set to one, the selected byte from the LUT is output on the least significant byte (bits 7 to 0) of the Cascade Output pins.

Pass LUT data to most significant output **BCR2[5]** **Default: 0**

This bit controls the output multiplexer. If this bit is set to one, the selected byte from the LUT is output on the most significant byte (bits 21 to 14) of the Cascade Output pins.

Look up prescaler **BCR2[4-0]** **Default: 00000**

The contents of these five bits control the amount of (arithmetic) right shift applied to the data, by the Look up Prescaler. Writing a numerical value between 0 and 16 (binary 10000) into these bits, will cause the data to be right-shifted by a corresponding number of places. For example, if the bit pattern 00101 is written to these five bit positions, a right shift of 5 places will occur. Writing any value outside the range (0 to 16) will result in undefined behaviour of the look up Prescaler.

10.7 Backend control register 3 (BCR3)

Normalizer control **BCR3[7-3]** **Default: 00000**

These five bits control the number of places, that the normaliser shifts the data to the right or to the left. This field is interpreted as a twos complement number. A positive number is taken to be a right shift. Any shift in the range -2 (11110) to +14 (01110) is legal. Any other value will cause the number zero to be output from the normaliser.

Enable dynamic normalization **BCR3[2]** **Default: 0**

If this bit is set to one, the normaliser will be controlled by bits 26 to 22 from the output of the look up table, instead of BCR3[7-3].

Feed LUT data to output adder **BCR3[1]** **Default: 0**

One of the inputs of the Output Adder can be either supplied by the Look up Table or forced to zero. Setting this control bit to zero selects zero. Setting this control bit to one selects bits 21 to 0 of the Look up Table.

Reserved **BCR3[0]** **Default: 0**

This location is reserved. The user should write zero to this location to maintain compatibility with future products. The value read from this location will be zero.

11. ELECTRICAL SPECIFICATION

11.1 DC electrical characteristics

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes (1,2)
V_{CC}	DC supply voltage	0		7.0	V	3
V_I, V_O	Voltage on any other pin	-1.0		V_{CC}+0.5	V	3
T_A	Temperature under bias	-40		85	°C	
T_{stg}	Storage temperature	-65		150	°C	
PDmax	Power dissipation			2.0	W	

Notes

1 All voltages are with respect to **GND**.
2 This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedence circuit. Unused inputs should be tied to an appropriate logic level such as **V_{CC}** or **GND**.

DC OPERATING CONDITIONS

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes (1)
V_{CC}	Supply Voltage	4.5	5.0	5.5	V	
V_{IH}	Input Logic '1' Voltage **CLK** Input Logic '1' Voltage other pins	4.0 2.0		V_{CC}+0.5 V_{CC}+0.5	V V	2 2
V_{IL}	Input Logic '0' Voltage **CLK** Input Logic '0' Voltage other pins	-0.5 -0.5		0.5 0.8	V V	2 2
TA	Ambient Operating Temperature	0		70	°C	3

Notes

1 All voltages are with respect to **GND**.
2 Input signal transients, up to 10ns wide, are permitted in the voltage ranges (**GND** - 0.5 V) to (**GND** - 1.0 V) and **V_{CC}** + 0.5 V to **V_{CC}** + 1.0 V.
3 400 linear ft/min transverse air flow.

DC CHARACTERISTICS

Symbol	Parameter	Min.	Typ.	Max.	Units	Notes (1,2)
V_{OH}	Output Logic '1' Voltage	2.4		V_{CC}	V	4
V_{OL}	Output Logic '0' Voltage	0		0.4	V	5
I_{IN}	Input leakage current(any input current)			± 10	µA	3
I_{OZ}	Off state output leakage current			± 10	µA	3
I_{DD}	Average power supply current			350	mA	

Notes

1 All voltages are with respect to **GND**.
2 Parameters measured over full voltage and temperature operating range.
3 $V_{CC} = V_{CC}$(max), GND ≤ V_{IN} ≤ V_{CC}
4 **IOut** ≤ -4.4 mA
5 **IOut** ≤ 4.4 mA

CAPACITANCE

Pin	Min.	Typ.	Max.	Units	Notes
CLK		12		pF	1,2
All other pins		5		pF	1,2

1 This parameter is supplied for engineering guidance and is not guaranteed.
2 TA= 25°C , F= 1 MHz.

11.2 Thermal Characteristics

PIN GRID ARRAY THERMAL CHARACTERISTICS

Symbol	Parameter	Min	Nom	Max	Units	Notes
θ JA	Junction to ambient thermal resistance			35	°C/W	1,2

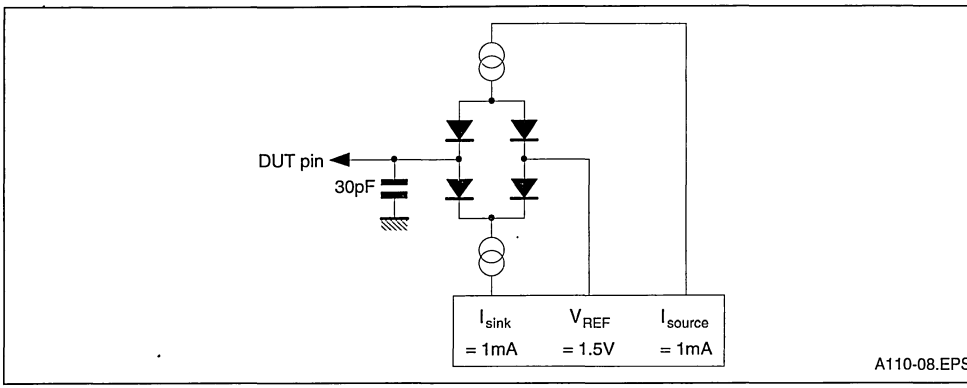
Notes
1 Measured at 400 linear ft/min transverse air flow.
2 This parameter is sampled and not 100% tested.

11.3 AC timing characteristics

AC test conditions

OUTPUT LOADS (except output turn-off tests) : 30pF for all outputs.

Figure 8 : Output Load (output turn-off tests)



TIMING REFERENCE LEVELS

Pin	Reference levels	Notes
INPUTS	0.8V, 2.0V	1
CLK	0.5V, 4.0V	
OUTPUTS	0.4V, 2.4V	2,3
OUTPUTS	±100mV change from previous steady output voltage	4

Notes
1 Except **CLK**.
2 Output continously driven.
3 Timings are tested using V_{OL}=0.8V and with a suitable allowance for the time taken for the output to fall from 0.8V to 0.4V.
4 Output turn-off tests.

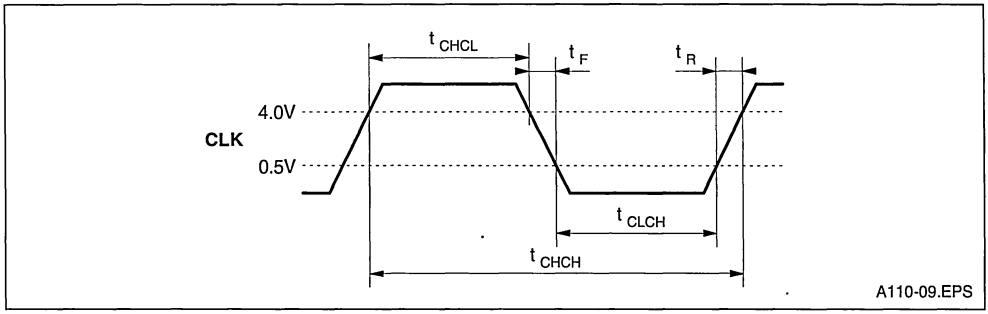
11.4 Timing diagrams

CLOCK REQUIREMENTS

Symbol	Parameter	Min	Typ.	Max	Units	Notes
t_{CHCL}	Clock Pulse High Width	20			ns	2
t_{CLCH}	Clock Pulse Low Width	20			ns	2
t_{CHCH}	Clock Period	50			ns	2
t_R	Clock rise time	0		50	ns	1
t_F	Clock fall time	0		50	ns	1

Notes
1 Clock input transitions should be monotonic between the input thresholds of 0.5 V and 4.0 V.
2 For Rev.A parts t_{CHCL}, t_{CLCH} and t_{CHCH} have **maximum** values of 50 000ns, 50 000ns and 100 000ns respectively. (A minimum clock frequency of 10kHz.)

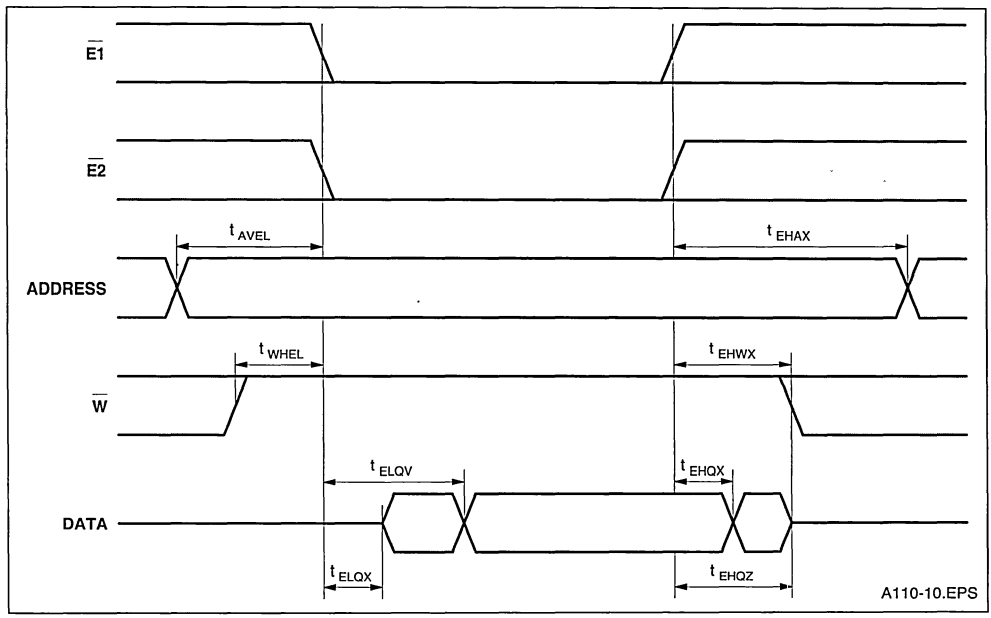
Figure 9



MICROPROCESSOR INTERFACE READ CYCLE

Symbol	Parameter	Min	Max	Units	Notes
t_{AVEL}	Address setup	0		ns	
t_{EHAX}	Address hold	0		ns	
t_{WHEL}	Read Command Setup	0		ns	
t_{EHWX}	Read Command Hold	0		ns	
t_{ELQX}	Output turn-on	0		ns	
t_{ELQV}	Read data access		100	ns	
t_{EHQX}	Read data hold	0		ns	
t_{EHQZ}	Output turn off		25	ns	

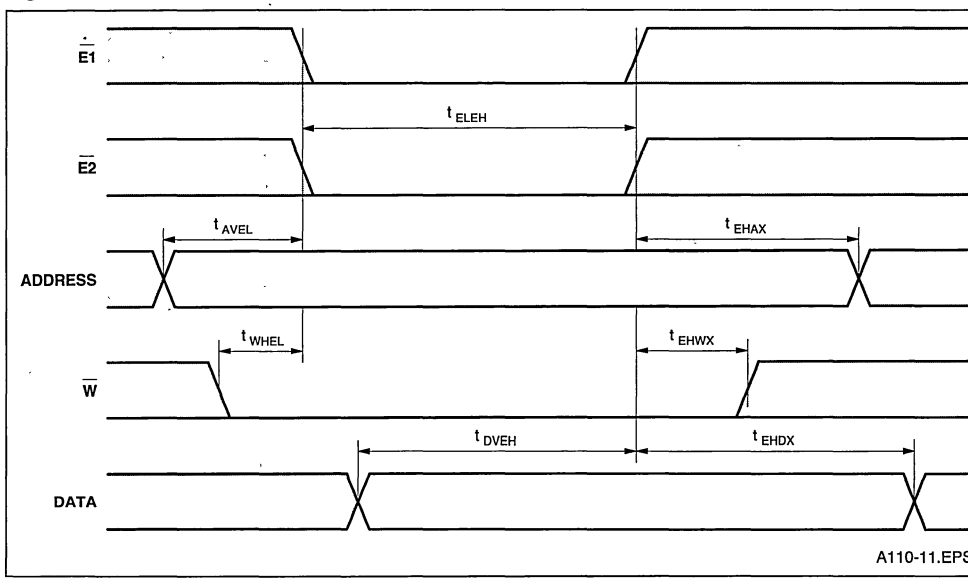
Figure 10



MICROPROCESSOR INTERFACE WRITE CYCLE

Symbol	Parameter	Min	Max	Units	Notes
t_{ELEH}	Enable Width Low	100		ns	
t_{AVEL}	Address setup	0		ns	
t_{EHAX}	Address hold	0		ns	
t_{WLEL}	Write Command Setup	0		ns	
t_{EHWX}	Write Command Hold	0		ns	
t_{DVEH}	Write data Set up	50		ns	
t_{EHDX}	Write data hold	0		ns	

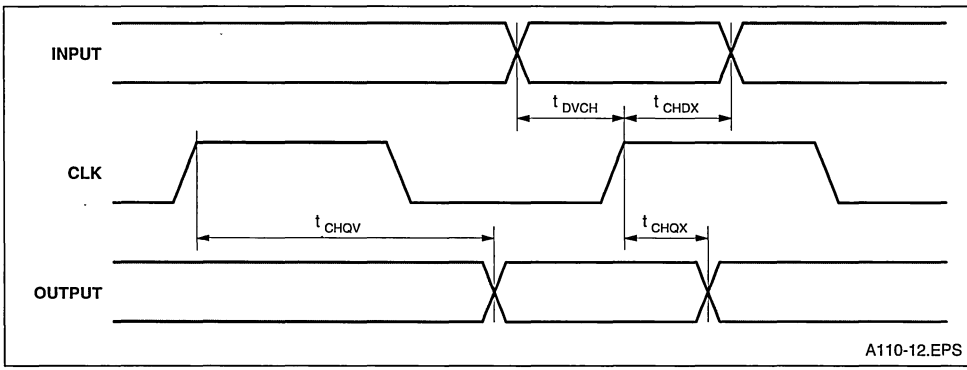
Figure 11



SYNCHRONOUS INPUT AND OUTPUT

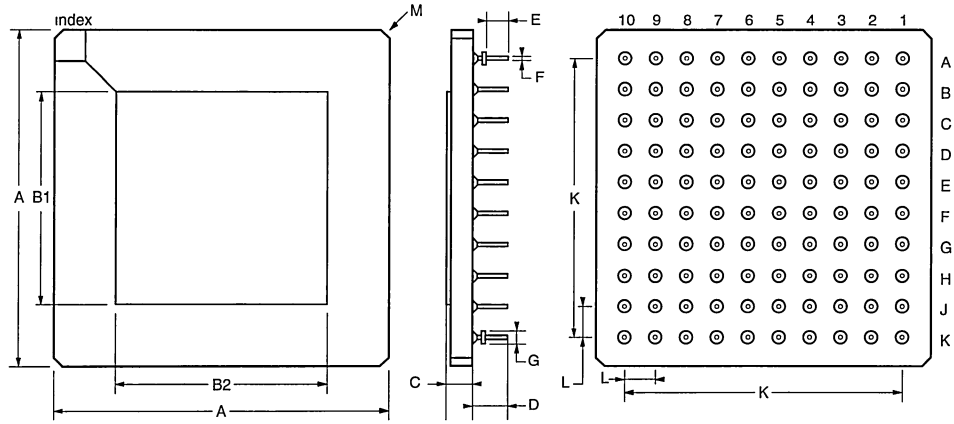
Symbol	Parameter	Min	Max	Units	Notes
t_{CHQV}	CLK high to Output Valid		40	ns	
t_{CHQX}	Output hold time after CLK	2		ns	
t_{DVCH}	Input setup time to CLK high	8		ns	
t_{CHDX}	Input hold time to CLK high	0		ns	

Figure 12



PACKAGE MECHANICAL DATA

100 PINS - GRID ARRAY PACKAGE



DIM	Millimetres		Inches		Notes
	Nom	Tol	Nom	Tol	
A	26.924	± 0.254	1.060	± 0.010	
B1	17.019	± 0.127	0.670	± 0.005	
B2	18.796	± 0.127	0.740	± 0.005	
C	2.456	± 0.278	0.097	± 0.011	
D	4.572	± 0.127	0.180	± 0.005	
E	3.302	± 0.127	0.130	± 0.005	
F	0.457	± 0.051	0.018	± 0.002	Pin diameter
G	1.143	± 0.127	0.045	± 0.005	Flange diameter
K	22.860	± 0.127	0.900	± 0.005	
L	2.540	± 0.127	0.100	± 0.005	
M	0.508		0.020		Chamfer

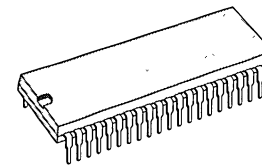
PGA100.TBL

SGS-THOMSON MICROELECTRONICS

STV8438

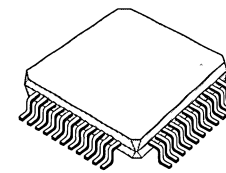
TRIPLE 8-BIT D/A CONVERTER

- 3 CHANNEL D/A CONVERTER
- 8-BIT RESOLUTION
- 70 MEGASAMPLES PER SECOND CONVERSION RATE
- AUXILIARY ANALOG R, G, B, SWITCHING CAPABILITIES
- SINGLE VOLTAGE +5V OPERATION
- ON-CHIP VOLTAGE REFERENCE
- VOLTAGE OUTPUT BUFFER AMPLIFIER
- TTL COMPATIBLE DIGITAL INPUTS
- BINARY INPUT ON ALL CHANNELS
- 2'S COMPLEMENT INPUT CAPABILITY ON TWO CHANNELS
- MONOLITHIC BIPOLAR
- 850 mW POWER DISSIPATION
- OPERATING TEMPERATURE RANGE $0^{o}C$ to + $70^{o}C$



SHRINK 42
(Plastic Package)

ORDER CODE : STV8438



PQFP 44
(Plastic Package)

ORDER CODE : STV8438CV

DESCRIPTION

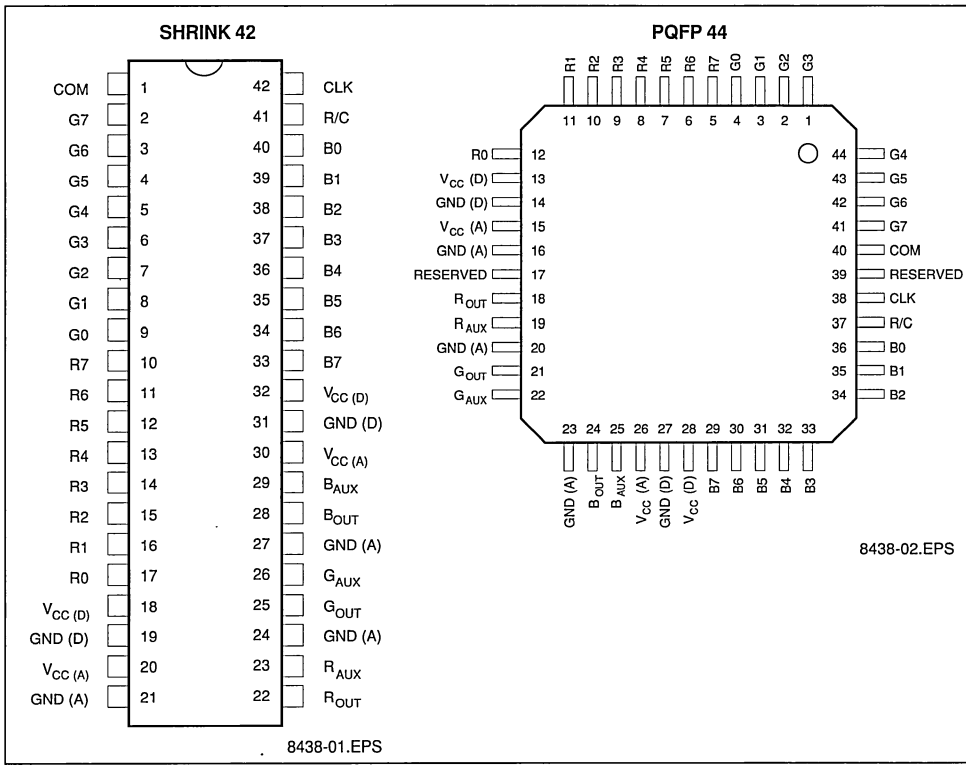
This Digital-to-Analog converter is a monolithic voltage output converter which can accept TTL-level digital input voltages.

The STV8438 contains three 8-bit D/A converters with a high performance on-chip voltage reference.

Internal analog multiplexing between the signals from the internal D/A converter and from auxiliary analog R, G, B signals is provided. Either binary or 2's Complement inputs are available for two of the three channels.

This device is particularly recommended for use in video processing applications with the capability of 70Msps data conversion rate with excellent linearity.

PIN CONNECTIONS



PIN ASSIGNMENT (SHRINK 42)

N° Pin Number	Symbol	Type	Function
1	COM	I	Analog switch selection
2 to 9	G < 0:7>	I	Digital input channel G
10 to 17	R < 0:7>	I	Digital input channel R
18 to 32	V_{CC} (D)	I	Digital power supply
19 to 31	GND (D)	I	Digital ground
20 to 30	V_{CC} (A)	I	Analog power supply
21	GND (A)	I	Analog R channel ground
22	R_{OUT}	O	Analog output, R channel
23	R_{AUX}	I	Auxiliary analog input, R channel
24	GND (A)	I	Analog G channel ground
25	G_{OUT}	O	Analog output, G channel
26	G_{AUX}	I	Auxiliary analog input, G channel
27	GND (A)	I	Analog B channel ground
28	B_{OUT}	O	Analog output, B channel
29	B_{AUX}	I	Auxiliary analog input, B channel
33 to 40	B < 0:7>	I	Digital input channel B
41	R/C	I	Binary or 2's complement selection
42	CLK	I	Clock input

8438-01.TBL

PIN DESCRIPTION

COM : Digital or analog inputs selection
This TTL input selects on the output stage the signal from the D/A converter or the signal from the external analog input. The three internal analog switches are activated by the COM signal.

COM = 0 connects auxiliary analog inputs to output amplifier
COM = 1 connects internal digital channel to output amplifier

G <0:7> : Digital input channel G
These TTL 8-Bit input data are sampled on the rising edge of the clock CLK. G_0 is the LSB and G_7 the MSB, coding is binary.

R <0:7>: Digital input channel R
These TTL 8-Bit input data are sampled on the rising edge of the clock CLK. R_0 is the LSB and R_7 the MSB. Coding is binary if the R/C input is high, coding is 2's complement if the R/C input is low.

B <0:7> : Digital input channel B
These TTL 8-Bit input data are sampled on the rising edge of the clock CLK. B_0 is the LSB and B_7 the MSB. Coding is binary if the R/C input is high, coding is 2's complement if the R/C input is low.

R/C : Binary/2's complement coding selection
This TTL input selects the coding type on R and B channels.

R/C = 0 selects 2's complement coding on R and B channels
R/C = 1 selects Binary coding on R and B channels

R_{AUX} : Auxiliary analog input, R channel
This analog input is connected to the output Rout through the output amplifier if the COM signal is low.

G_{AUX} : Auxiliary analog input, G channel
This analog input is connected to the output G_{OUT} through the output amplfier if the COM signal is low.

B_{AUX} : Auxiliary analog input, B channel
This analog input is connected to the output B_{OUT} through the output amplifier if the COM signal is low.

R_{OUT} : Analog output, R channel
This voltage analog output corresponds to the digital channel R if the COM signal is high or to the auxiliary analog input R_{AUX} it the COM signal is low.

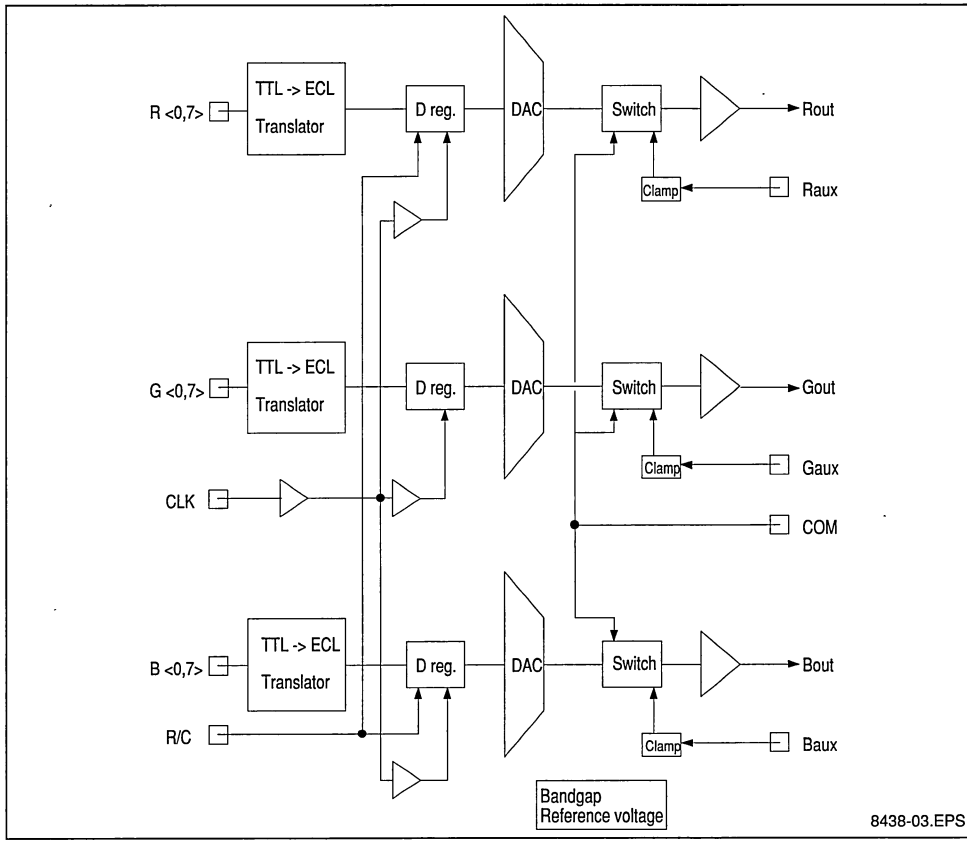
GOUT : Analog output, G channel
This voltage analog output corresponds to the digital channel G if the COM signal is high or to the auxiliary analog input G_{AUX} if the COM signal is low.

BOUT : Analog output, B channel
This voltage analog output corresponds to the digital channel B if the COM signal is high or to the auxiliary analog input B_{AUX} if the COM signal is low.

CLK : Clock signal
The digital inputs are sampled on the rising edge of this TTL input signal.

V_{CC} (A) / GND (A) : Analog power supply
V_{CC} (D) / GND (D) : Digital power supply.

BLOCK DIAGRAM



CIRCUIT DESCRIPTION

The STV8438 is designed with 3 identical D/A converters. Each D/A converter is constituted of two 4-bit DACs separated by a current divider the elementary DAC is composed of multiple identical current switches supplied with the same current allowing high speed conversion rate.

DIGITAL INPUT CHANNELS

The STV8438 supports binary coding on the 3 R, G, B, input channels when R/C pin is high. When R/C pin is low, a 2's complement coding is applied to the R and B channels this provides the capability to use the STV8438 with luminance and chrominance coded signal ; the luminance signal (usually called Y) being applied to the G channel, the chrominance signals (called U, V) being applied respectively to the R and B channels.

The input range on Y signal is 0/255 and the input range on both U, V signals is -128/+127. Whatever binary coding or 2's complement coding the output voltage is in the range of 1.685V for the lowest code to 3.315V for the highest code.

ANALOG INPUT CHANNELS

The STV8438 provides the capability to switch the output voltage from signals coming from the digital channels or from signal coming from auxiliary analog inputs. When COM signal is low, the auxiliary analog signals are connected to the output amplifier internaly clamped to the 16th digital step. When COM signal is high, the digital inputs after D/A conversion are connected to the output amplifier.

ANALOG OUTPUTS

The output voltage amplifiers have an output range of 1.685V to 3.315V. The 1.685V corresponds to the binary code Ø (R/C = 1) or to the 2's complement code -128 (R/C = 0). The 3.315V corresponds to the maximum value on the digital code 255 if R/C = 0, +127 if R/C = 1.

The STV8438 provides a step if 6.39mV per LSB. Using the analog input signal (COM = 0), the output amplifier has a gain of 2.

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{CC}	Supply Voltage	8	V
P_{tot}	Power Dissipation	1.8	W
T_{oper}	Operating Temperature	- 40 to 85	°C
T_{stg}	Storage Temperature	- 55 to 150	°C

8438-02.TBL

DC ELECTRICAL CHARACTERISTICS (Temperature 0 to 70°C, $V_{CC} \pm 5\%$)

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{CC}	Supply Voltage	4.75	5	5.25	V
I_{CC}	Supply Current		170		mA
	Resolution			8	Bit
V_{OUTF}	Full Scale Output Voltage		3.315		V
V_{OUTZ}	Zero Scale Output Voltage		1.685		V
DL	Differential Linearity Error			± 0.5	LSB
IL	Integral Linearity Error			1	LSB
	Gain Conversion Error between RGB			± 2	%
P_D	Power Dissipation		850		mW

8438-03.TBL

AC ELECTRICAL CHARACTERISTICS (Temperature 0 to 70°C, $V_{CC} \pm 5\%$) (continued)

Symbol	Parameter	Min.	Typ.	Max.	Unit
ANALOG OUTPUTS					
	Maximum Data Conversion Rate	70			Msps
t_S	Settling Time Figure 1 Figure 2			 14 28.5	ns
	Monotonicity	Guaranteed			
	Glitch Energy			80	pVs
t_{PD}	Propagation Delay (Figures 1 and 2)			4	ns
	Crosstalk between Any Outputs (f_{CLK} = 25MHz-input voltage.7V_{pp})	50			dB
	Crosstalk between any outputs when auxiliary analog inputs are selected (f_{CLK} = 25MHz .7V_{pp})	50			dB
R_{LOAD}	Output Load (AC coupled - see typical application diagram)	100	150		Ω
V_{OUT}	Output Voltage Range (on 150Ω AC coupled)		1.63		V_{PP}
AUXILIARY ANALOG RGB INPUTS					
t_{SW}	Switching-time DAC/Analog Input (Figure 3)			5	ns
	Black Level Clamp Error			± 2.5	%
	Crosstalk between Any Outputs (f = 5MHz-input voltage.7V_{pp})	50			dB
	Crosstalk between RGB Analog Inputs and D/A Outputs (f = 5MHz-input voltage.7Vpp)	50			dB

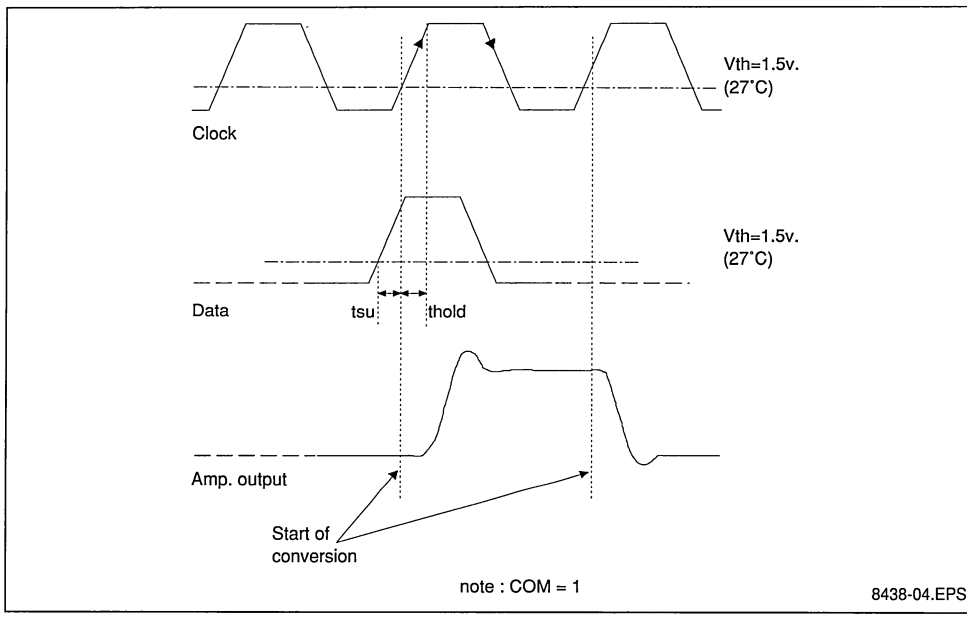
8438-04.TBL

AC ELECTRICAL CHARACTERISTICS (Temperature 0 to 70°C, $V_{CC} \pm 5\%$) (continued)

Symbol	Parameter	Min.	Typ.	Max.	Unit
ANALOG OUTPUTS FROM ANALOG INPUTS					
G	Voltage gain at f=1MHz (input voltage .7Vpp)		2.0		
BNa	Band-width (-3dB)	100			MHz
	Slew-rate (inp. pulse 0.7Vpp)	120	150		V/µs
	Harmonic distortion rate at 1MHz			0.2	%
DIGITAL INPUTS					
V_{IN}	Input Voltage High Level	2			V
V_{IL}	Input Voltage Low Level			0.8	V
I_{IH}	Input Current High Level			10	µA
I_{IL}	Inpur Current Low Level	-500			µA
SWITCHING CHARACTERISTICS					
FCK	Clock Rate		100		MHz
	Clock Duty-cycle Rate		50		%
t_{CKR}	Clock Rise-time (10% - 90%)			3.5	ns
t_{CKF}	Clock Fall-time (90% - 10%)			3.5	ns
t_{SU}	Data Set-up Time to CLK	2.5			ns
t_{HOLD}	Data Hold-time from CLK	2.5			ns
t_D	Data Conversion Delay		1		cycle

8438-05.TBL

INPUT TIMING DIAGRAM



SETTLING TIME MEASUREMENTS

Figure 1

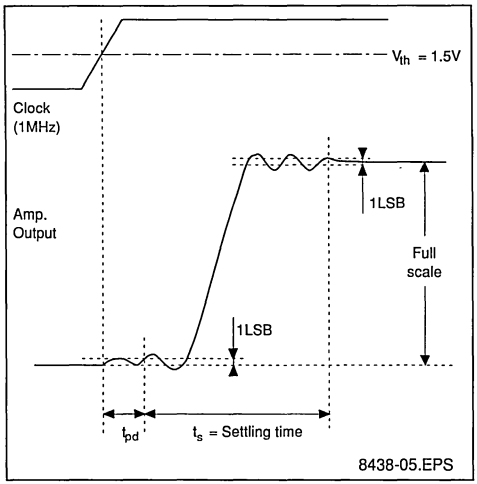
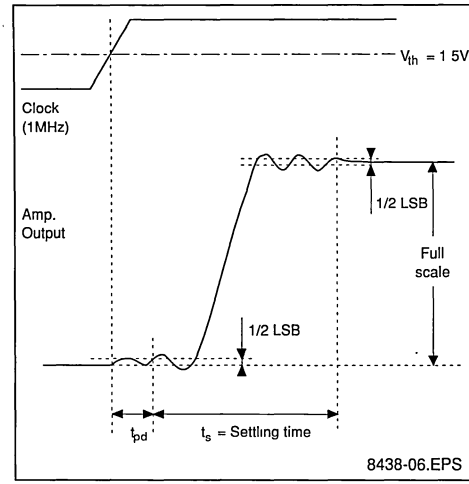
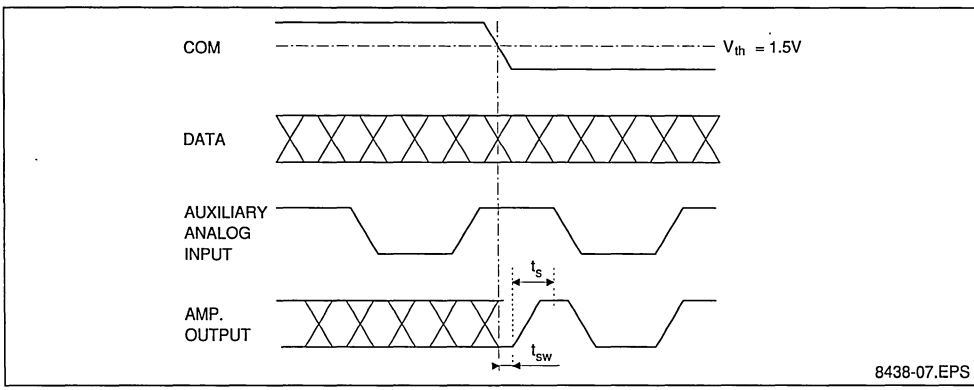


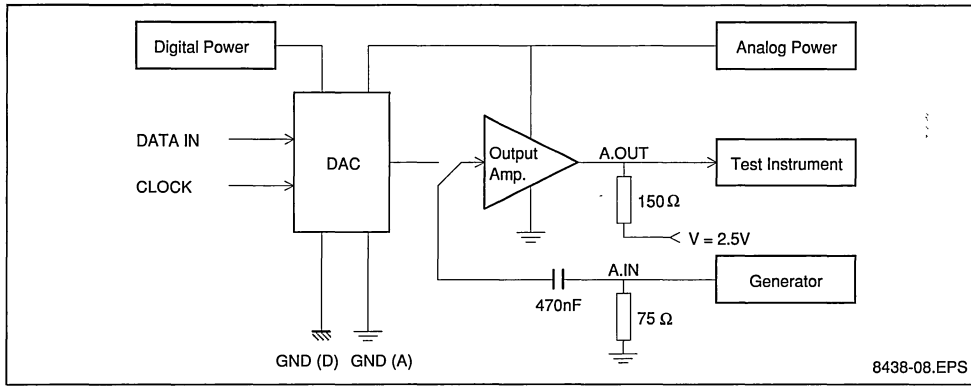
Figure 2



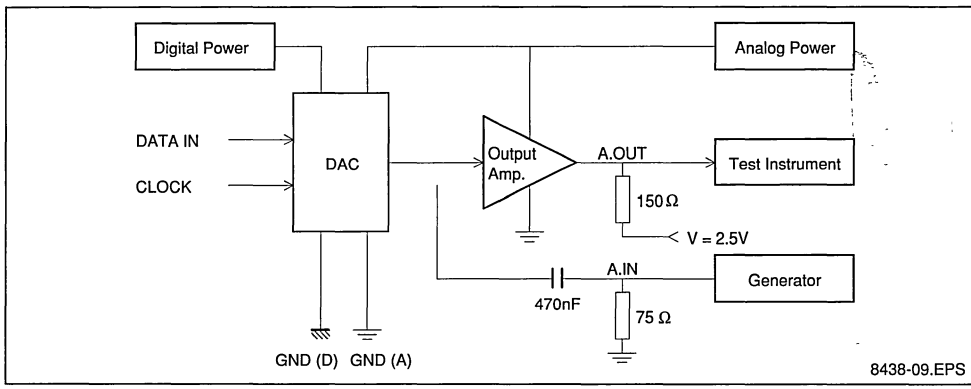
SWITCHING TIME DAC/AUXILIARY ANALOG INPUT MEASUREMENT



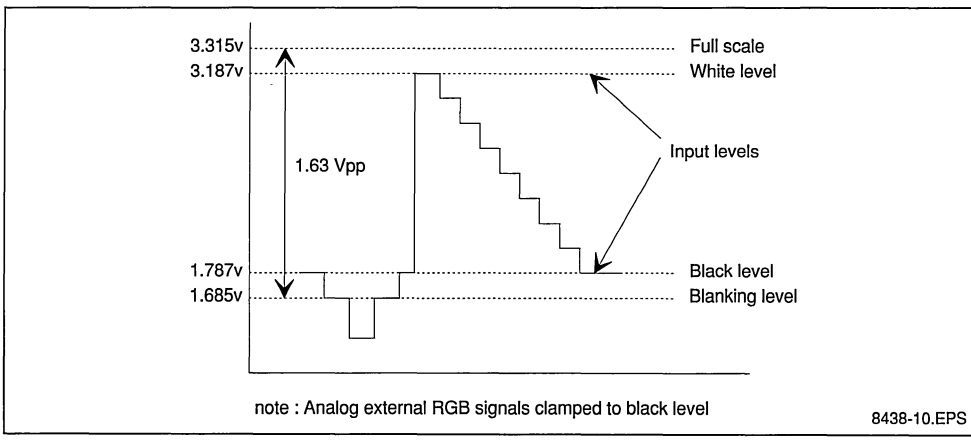
ANALOG-TEST SCHEMATICS



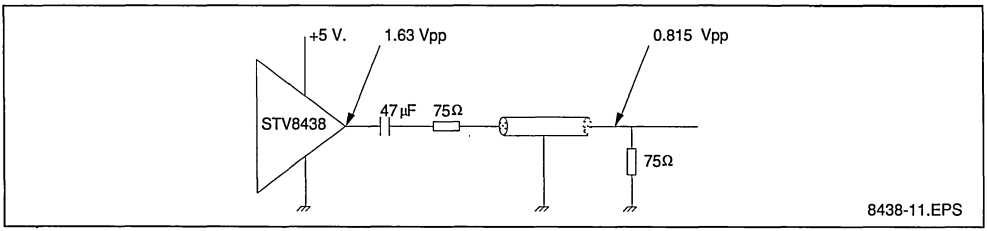
DAC-TEST SCHEMATICS



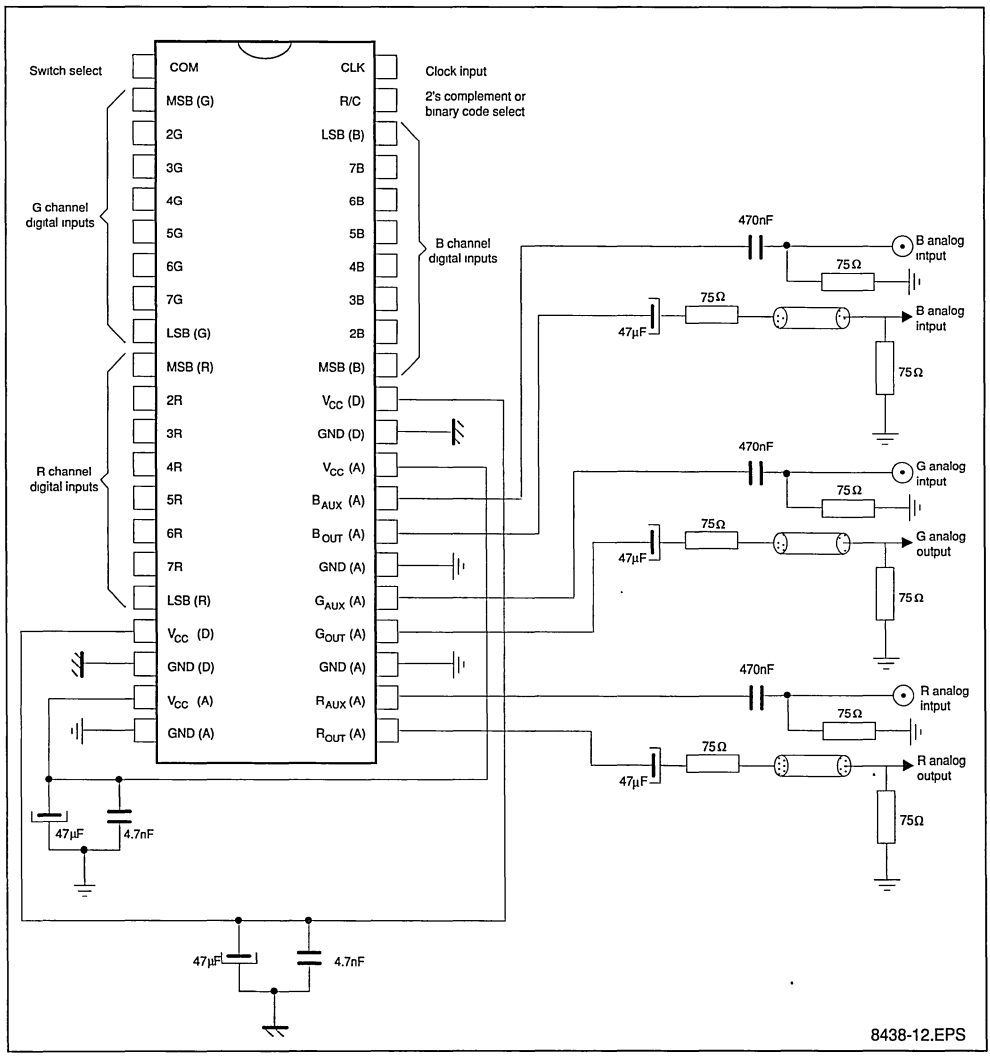
VOLTAGE AT BUFFER OUTPUT



OUTPUT 75Ω MATCHING

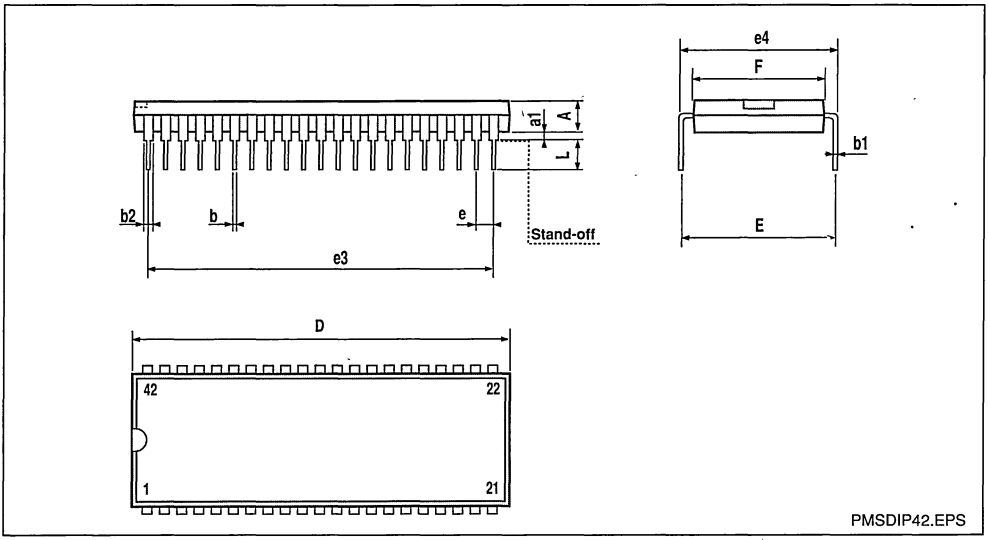


TYPICAL APPLICATION



PACKAGE MECHANICAL DATA

42 PINS - PLASTIC SHRINK DIP

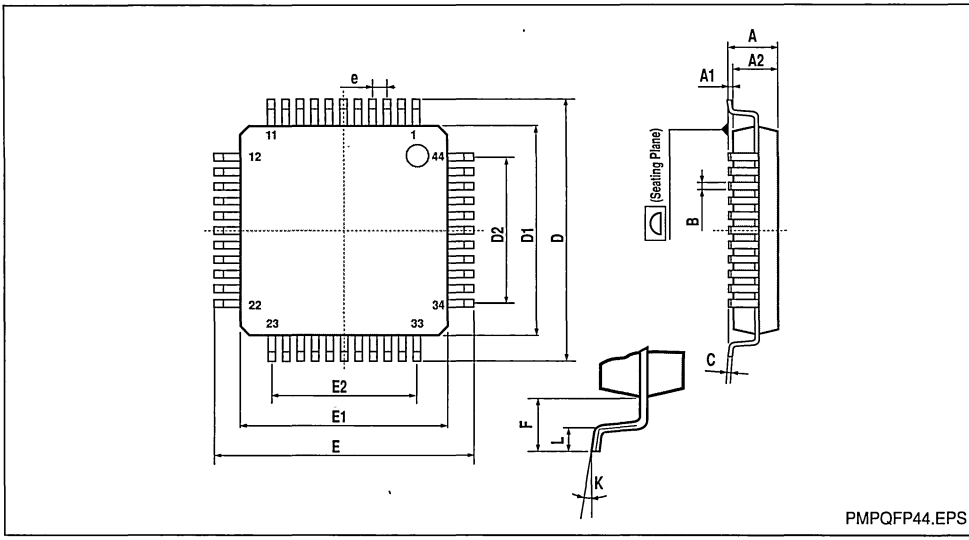


Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A			4.83			0.190
a1	0.51			0.020		
b	0.38	0.46	0.56	0.015	0.018	0.022
b1	0.20	0.25	0.30	0.008	0.010	0.012
b2	0.76	1.02	1.27	0.030	0.040	0.050
b3		0.75			0.030	
D	36.70	36.83	36.96	1.445	1.450	1.455
E		15.24			0.600	
e	1.778			0.070		
e3	35.56			1.400		
e4	15.24			0.600		0.625
F	13.46	13.72	13.97	0.530	0.540	0.550
L	3.05		3.43	0.120		0.135

SDIP42B.TBL

PACKAGE MECHANICAL DATA

44 PINS - PLASTIC QUAD FLAT PACK



Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A			3.40			0.134
A1	0.25			0.01		
A2	2.55	2.80	3.05	0.10	0.11	0.12
B	0.35		0.50	0.014		0.020
C	0.13		0.23	0.005		0.009
D	16.95	17.20	17.45	0.667	0.677	0.687
D1	13.90	14.00	14.10	0.547	0.551	0.555
D2		10.00			0.394	
e		1.00			0.039	
E	16.95	17.20	17.45	0.667	0.677	0.687
E1	13.90	14.00	14.10	0.547	0.551	0.555
E2		10.00			0.394	
F		1.60			0.063	
K	0^0 (min.), 7^0 (max.)					
L	0.65	0.80	0.95	0.025	0.031	0.037

PQFP44.TBL

DATASHEETS

EVALUATION BOARDS

SGS-THOMSON MICROELECTRONICS

EVAL3208

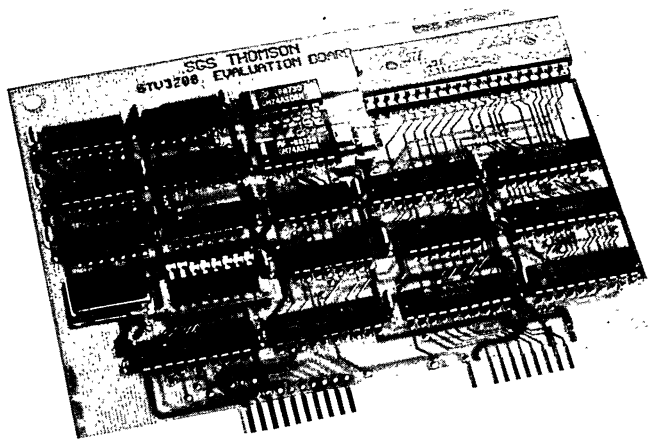
STV3208 EVALUATION BOARD

ADVANCE DATA

- IBM - PC XT, AT COMPATIBLE HALF CARD
- ALLOWS THE PC DIRECT ACCESS TO ALL THE PORTS OF AN STV3208
- MULTIPLE BOARDS MAY BE PLUGGED INTO A SINGLE PC, ALLOWING THE DIFFERENT FUNCTIONS AVAILABLE ON THE STV3208 TO BE PERFORMED SIMULTANEOUSLY
- DIRECTLY COMPLEMENTS THE STV3208 C MODEL

APPLICATIONS

- ACCELERATION OF PC BASED IMAGE PROCESSING OPERATIONS AND THE STV3208 C MODEL
- SYSTEM PROTOTYPING
- ENGINEER FAMILIARIZATION WITH THE STV3208



SGS-THOMSON MICROELECTRONICS

EVAL3220

STV3220 EVALUATION BOARD

ADVANCE DATA

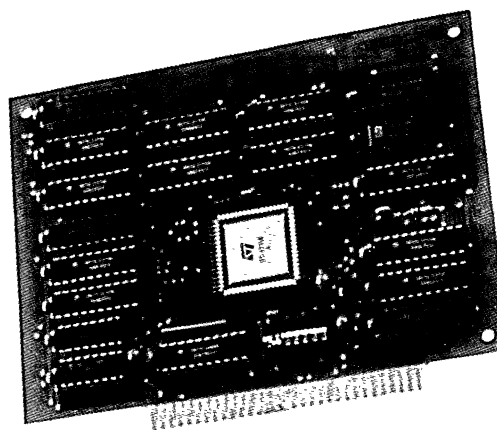
- SLAVE BOARD (IBM - PC XT or AT compatible)
- BLOCK SIZE : 8 x 4n, 16 x 4n
- MAXIMUM DISPLACEMENT +7/-8 PIXELS HORIZONTALLY AND VERTICALLY
- COMPUTATION OF THE MOTION VECTOR AND MINIMUM DISTORTION
- RANDOM ACCESS TO THE 256 DISTORTIONS
- 8-BIT UNSIGNED INPUT PIXEL
- 4-BIT 2'S COMPLEMENT HORIZONTAL DISPLACEMENT
- 4-BIT 2'S COMPLEMENT VERTICAL DISPLACEMENT
- 16-BIT UNSIGNED DISTORTION VALUES
- BLOCK EDITOR
- DEBUGGING PROGRAM
- TIMING GENERALLY CONTROLLED USING THE PC BUS

REQUIRED EQUIPMENT

- IBM PC RACK : XT or AT
- A MONOCHROME OR A COLOR GRAPHIC CAD DRIVING A MONITOR

SOFTWARE DESCRIPTION

The EVAL3220 package contains software, tools and documentation to help the programmer take full advantage of all the user interface management features. The EVAL3220 is a menu-driven software package enveloped to assist the programmer of the STi3220 chip. The software processes 8 x 4n or 16 x 4n block size. Blocks can be edited, loaded and saved using menu functions. The user can check step by step the STi3220 input pixel's block or run a part or a complete block ending with a break point. A selected distortion or the distortion set can be displayed. A software simulation of the chip is also available upon request.



SGS-THOMSON MICROELECTRONICS

EVALA110

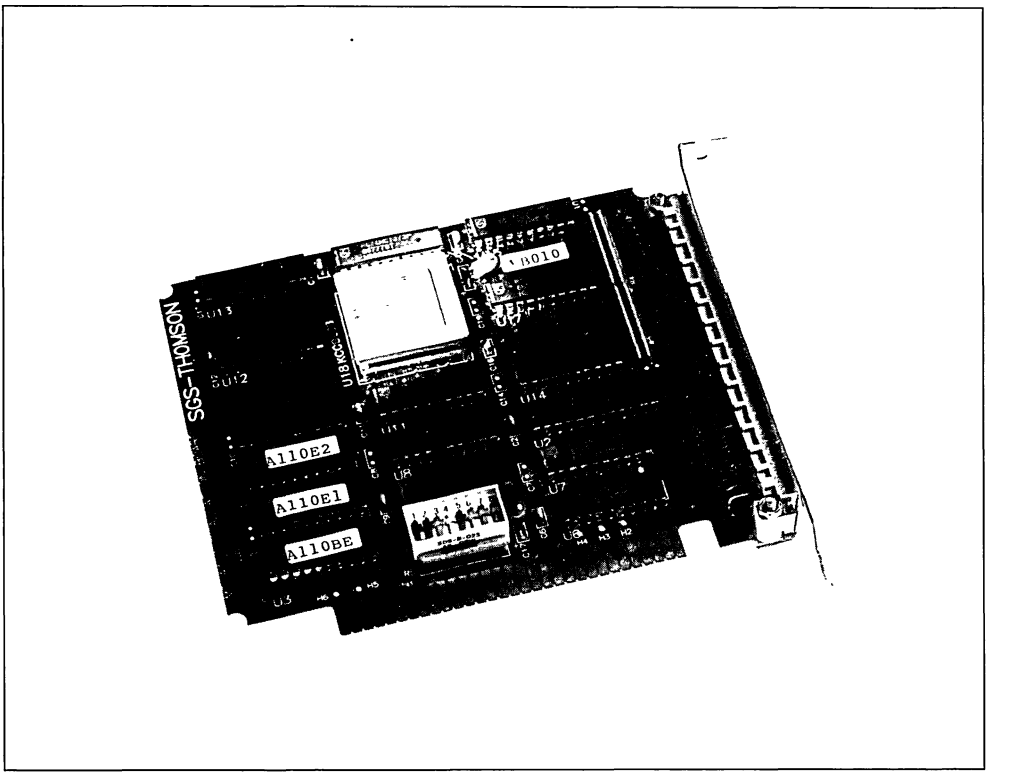
IMSA110 EVALUATION BOARD

ADVANCE DATA

- IBM - PC XT, AT COMPATIBLE HALF CARD
- ALLOWS THE PC DIRECT ACCESS TO ALL THE PORTS OF AN IMSA110
- EXTERNAL CONNECTIONS ARE AVAILABLE FOR PROCESSING REAL TIME DATA, AND INTERACTING INTO SYSTEMS
- MULTIPLE BOARDS MAY BE PLUGGED INTO A SINGLE PC, ALLOWING IMSA110 CASCADES TO BE BUILT
- DIRECTLY COMPLEMENTS THE IMSA110 C MODEL

APPLICATIONS

- ACCELERATION OF PC BASED IMAGE PROCESSING OPERATIONS AND THE IMSA110 C MODEL
- ENGINEER FAMILIARIZATION WITH THE IMSA110
- SYSTEM PROTOTYPING
- EDGE AND FEATURE DETECTION
- DATA TRANSFORMATION AND HISTOGRAM EQUALIZATION
- COMPUTER VISION
- TEMPLATE MATCHING
- 1-D OR 2-D INTERPOLATION
- OCR ACCELERATION



SGS-THOMSON MICROELECTRONICS

EVALA121

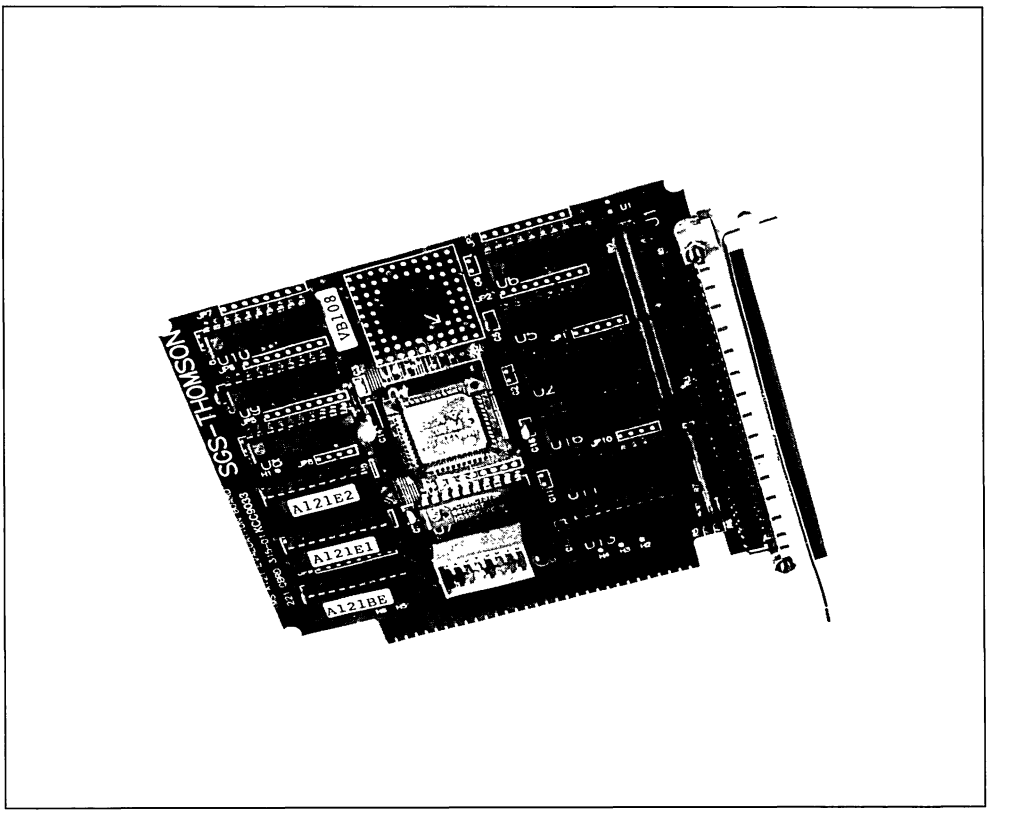
IMSA121 EVALUATION BOARD

ADVANCE DATA

- IBM - PC XT, AT COMPATIBLE HALF CARD
- ALLOWS THE PC DIRECT ACCESS TO ALL THE PORTS OF AN IMSA121
- EXTERNAL CONNECTIONS ARE AVAILABLE FOR PROCESSING REAL TIME DATA, AND INTERACTING INTO SYSTEMS
- MULTIPLE BOARDS MAY BE PLUGGED INTO A SINGLE PC, ALLOWING THE DIFFERENT FUNCTIONS AVAILABLE ON THE IMSA121 TO BE PERFORMED SIMULTANEOUSLY
- DIRECTLY COMPLEMENTS THE IMSA121 C MODEL

APPLICATIONS

- ACCELERATION OF PC BASED IMAGE PROCESSING OPERATIONS AND THE IMSA121 C MODEL
- SYSTEM PROTOTYPING
- ENGINEER FAMILIARIZATION WITH THE IMSA121



July 1992

1/1

APPLICATION NOTES

SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

DIGITAL FILTERING WITH THE IMSA100

SUMMARY

I. INTRODUCTION

When an analogue signal is sampled in time, the sampled signal is referred to as a discrete-time signal. If each sample in this discrete-time signal is also quantised in amplitude, (e.g. represented by an arbitrary n-bit number), then it is usually referred to as a digital signal. In the subject of digital filtering it is these types of signals which are processed and operated on. The fact that the digital signals are quantised both in time and amplitude gives one greater control over the processing as compared to analogue signal processing.

In these application notes the concept of the digital filtering is first introduced. This is done by starting from a simple RC analogue filter and deriving a corresponding digital filter. The classification of digital filters is then summarized, followed by giving a summary of techniques applicable to filter design using the IMSA100 device.

II. FROM ANALOGUE TO DIGITAL

Figure 1a shows a simple first-order RC filter. The simple differential equation describing this circuit in terms of its input and output voltages is:

$$v_o(t) + RC\frac{dv_o(t)}{dt} = v_i(t) \qquad (1)$$

where $v_0(t)$ and $v_i(t)$ are analogue output and input voltage waveforms. In the analogue world both input and output voltages are continuous-time waveforms and the complexity of the solution would depend on the input voltage function $v_i(t)$. Given an input waveform $v_i(t)$, the solution can be obtained using:

(i) Standard mathematical techniques which solve the differential equation and obtain the output waveform in closed form.

(ii) Numerical techniques which calculate the approximate output waveform in a digital computer. This would necessitate the sampling of the input and output waveforms.

If T is sufficiently small then the derivative $\frac{dv_o(t)}{dt}$ at time t=NT can be approximated by:

$$\frac{dv_o(nT)}{dt} \approx \frac{v_o(nT) - v_o[(n-1)T]}{T} \qquad (2)$$

substituting this in equation (1) we obtain:

$$v_o(nT) + \frac{RC}{T}v_o(nT) - \frac{RC}{T}v_o[(n-1)T] = v_i(nT) \qquad (3a)$$

Equation (3a) is a linear difference equation that approximates the differential equation (1). Equation (3a) can be rewritten as:

$$v_o(nT) = \frac{1}{1+(RC/T)}v_i(nT) + \frac{(RC/T)}{1+(RC/T)}v_o[(n-1)T] \qquad (3b)$$

This is now a recursion formula in which the present input sample and the previous output sample are used to calculate the present output sample. The notation can be simplified to:

$$v_o(n) = b_0\, v_i(n) + a_1\, v_o(n-1) \qquad (4a)$$

where $b_0 = \frac{1}{1+(RC/T)}$ and $a_1 = \frac{(RC/T)}{1+(RC/T)}$

The signal-flow diagram for this filter is shown in Figure 1b. The block labelled 'D' represents a delay equal to one sampling period T. In digital filter notations a delay of n sampling periods is usually denoted by z^{-n}. Therefore a delay of one sampling period can be represented by z^{-1}.

It is important to note that a common element in all filter structures is the concept of storage. In the analogue RC filter (Figure 1a) the storage is present in the form of a capacitor and in its digital equivalent (Figure 1b) the storage takes the form of a delay stage. In fact the storage element is the essential ingredient for any filter whether analogue or a digital. This is because filters are used to operate on the signal 'changes' and as such they need to have some knowledge of the history of the signal to allow them to perform their function.

An important characteristic feature of any filters is its so called 'impulse response'. This is defined as the output waveform of the filter when a unity impulse is applied to the input. Using equation (4a) and assuming a unity impulse as the input waveform i.e.

$$v_i(0) = 1$$
$$v_i(n) = 0 \text{ for } n > 0$$

then the output sequence would be:

$$b_0,\ a_1b_0,\ a_1^2b_0,\ldots\ldots,a_1^nb_0,\ldots\ldots\ldots$$

or in short $v_o(n) = a_1^n\, b_0$

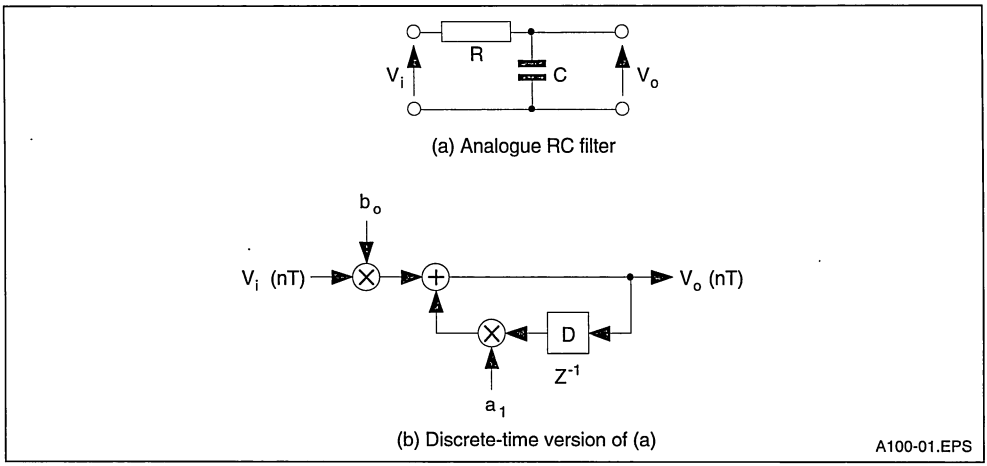
It should be noted that the above impulse response has, in theory, infinite length. This is due to the recursive nature of this particular filter structure. This types of filters are often referred to as infinite-impulse-response (IIR) filters.

An alternative way of looking at the filter in this example is to use equation (4a) in successive substitutions i.e.

$$\begin{aligned} v_o(n) &= b_0\, v_i(n) + a_1\, v_o(n-1) \qquad (4b)\\ &= b_0\, v_i(n) + a_1\,[\, b_0\, v_i(n-1) + a_1\, v_o(n-2)\,]\\ &= b_0 v_i(n) + a_1 b_0 v_i(n-1)\\ &\quad + a_1^2\,[\, b_0 v_i(n-2) + a_1 v_o(n-3)\,]\\ &= .....\\ &= .....\\ &= b_0\, v_i(n) + a_1\, b_0\, v_i(n-1) + a_1^2\, b_0\, v_i(n-2)\\ &\quad + a_1^3\, b_0\, v_i(n-3) + ... \end{aligned}$$

Equation (4b) expresses the output waveform as a linear combination of input samples only, but this involves infinite number of input samples. Notice also that the coefficients b_0 and a_1 have positive values less than unity (R and C are assumed to be finite and non-zero). This means that in equation (4b) the coefficients decrease for older input samples. It may therefore be reasonable to assume that these coefficients approximate to zero beyond a certain point. In this way only a finite number of terms would be involved in equation (4b), or in other words, the infinite impulse response is approximated by a finite impulse response since it decays rapidly to zero. This modified filter with its finite duration impulse response falls in the category of FIR (Finite-Impulse Response) filters. In the next section these concepts are generalized.

Figure 1 : Analogue RC Filter and its Discrete-time Equivalent



III. DIGITAL FILTER CLASSIFICATIONS

Linear difference equations, similar to equation (4a & 4b) are the basis for the theory of digital filters. The general difference equation can be expressed as:

$$y(n) + \sum_{m=1}^{M} a_m y(n-m) = \sum_{k=0}^{N} b_k x(n-k) \qquad (5)$$

Where the x and y sequences are the input and the output of the filter and a_m's and b_k's are the coefficients of the filter.

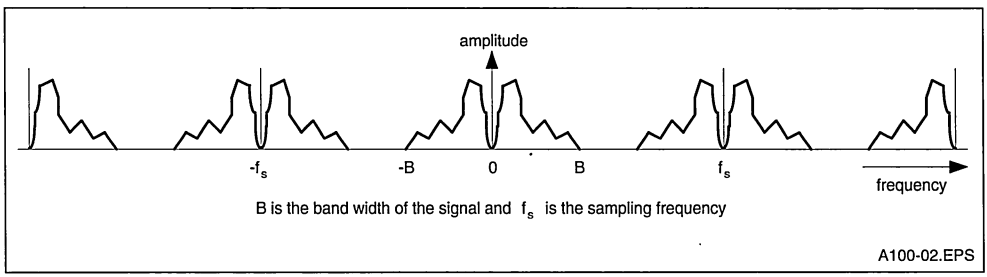
As mentioned earlier the notation z^{-1} is often used to denote a delay equal to one sampling period. In the theory of the dicrete-time signals, the concept of z has been developed further and is referred to as the z-transform. This is a discrete-time version of the well known Laplace transform (sometimes referred to as the s-transform) which is mainly used for dealing with continuous signals. In the s-domain a delay of T seconds corresponds to e^{-sT}. Therefore the two variables s and z are related by:

$$z^{-1} = e^{-sT} \qquad (6)$$

where T is the sampling period.

In the s-domain the spectrum of a signal with a bandwidth B and sampled at a frequency f_s, is periodic with a period equal to f_s. This is depicted in Figure 2. This periodicity in the spectrum of a sampled signal is the basic reason behind the Nyquist criterion which requires a minimum sampling frequency of twice the signal bandwidth (i.e. $f_{smin} = 2 \times B$), in order to avoid aliasing effects.

Figure 2 : Spectrum of a Sampled Signal



Equation (6) allows a mapping between the two domains. Part of the imaginary axis between $-\frac{f_s}{2}$ to $+\frac{f_s}{2}$, in the s-plane, is mapped into a unit circle in the z-domain as shown in Figure 3. The fact that the imaginary axis in the s-plane is mapped onto a circle is a consequence of the periodic nature of the spectrum. As shown in Figure 3, the left-hand half of the s-plane (between $-\frac{f_s}{2}$ and $+\frac{f_s}{2}$) is mapped onto the inside of the unit circle, while the right-hand half is mapped onto the outside of the circle.

As in the analogue design (s-domain) where a pole in the wrong place, i.e. in the right-half plane, indicates instability, in the case of discrete-time signals (z-domain) a pole outside the unit circle causes instabilities. In both cases zeroes can be anywhere.

Using the z-transform notation, the general linear equation (5) can be expressed as:

$$Y(z)\,(1 + \sum_{m=1}^{M} a_m z^{-m}) = X(z) \sum_{k=0}^{N} b_k z^{-k} \qquad (7)$$

Where X(z) and Y(z) are the z-transforms of the input and output waveforms. The discrete-time (or digital) transfer function of the general filter is thus given by:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{m=1}^{M} a_m z^{-m}} \qquad (8)$$

In terms of realization, digital filters are classified into nonrecursive and recursive types. The nonrecursive structure contains only feed-forward paths and as such all the a_m terms (equation (8)) are zero. This means that for the nonrecursive filters the output is a sum of linearly weighted present and a number of past samples of the input signal as shown in Figure 4. Referring to equation (8), for the nonrecursive filters the transfer function has only zeroes and as such is always stable.

In the recursive filters on the other hand some or all of the a_m terms are non-zero resulting in the presence of both poles and zeroes in the transfer function. Figure 5 shows the general recursive filter structure. Figure 6 shows an alternative structure for the same transfer function with a reduced number of delay stages.

Digital filters are also classified in terms of their impulse responses. In this classification those filters with a finite duration impulse response are referred to as FIR filters and those with an infinite duration impulse response are called IIR filters. The simplest FIR filter realization is in the nonrecursive form. For example in Figure 4, if a unit impulse is clocked through the filter, the sequence,

$$b_0, b_1, b_2, \ldots b_N, 0, 0, 0, 0, 0, 0, \ldots 0, 0, 0 \qquad (9)$$

will be output. Notice that the response consists of a sequence of samples corresponding to the filter coefficients followed by zeroes, i.e. the nonrecursive structure is an FIR filter. On the other hand the impulse response of the recursive structure (Figures 5 & 6), because of the feedback paths, is infinite in duration, making the configuration an IIR filter.

Figure 3 : Relation Ship between the s-domain and the z-domain

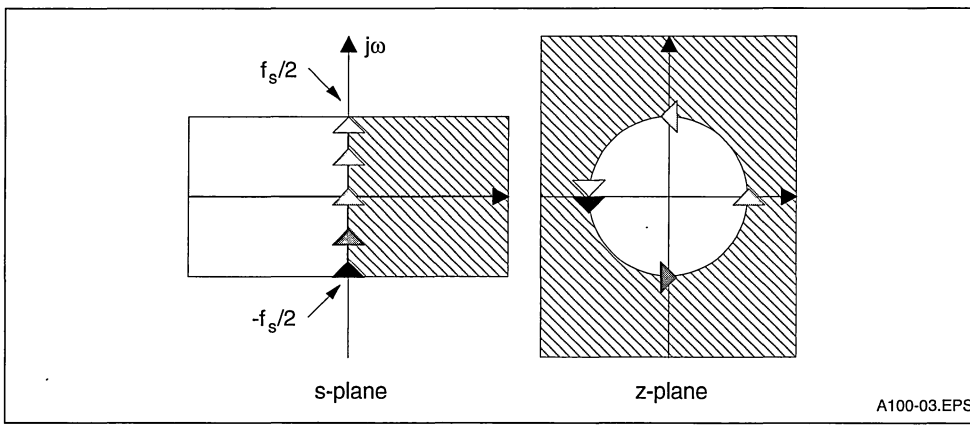


Figure 4 : Nonrecursive Digital Filter Structure

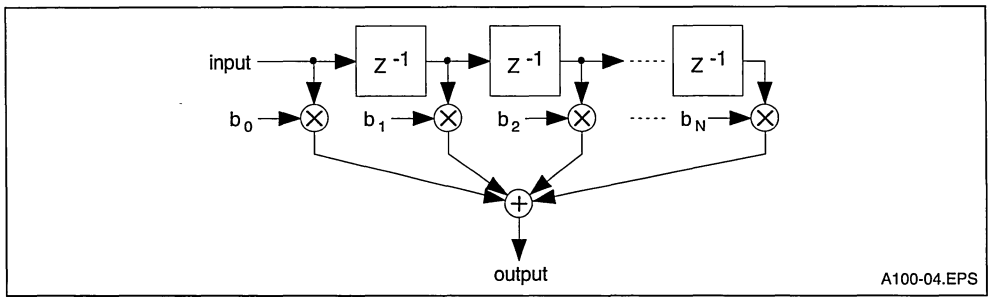


Figure 5 : Recursive (IIR) Digital Filter Structure

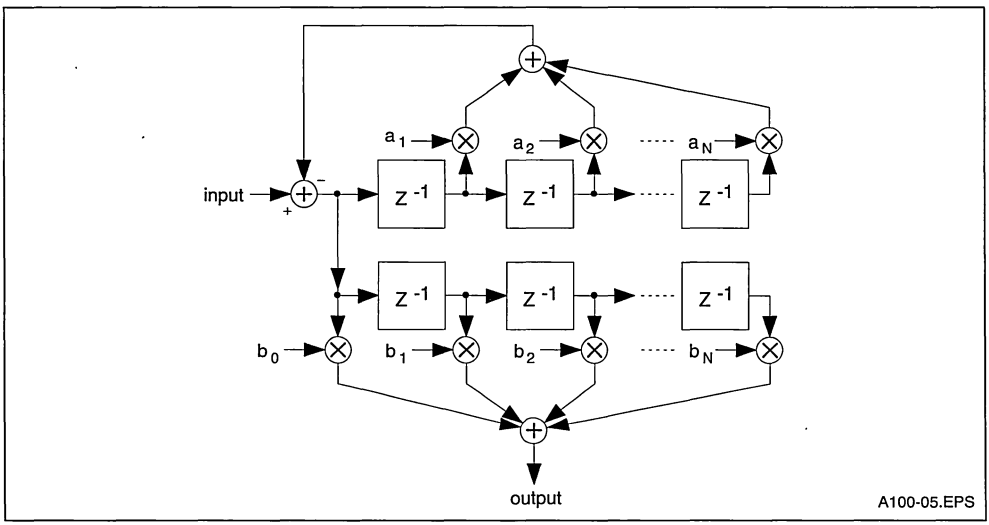
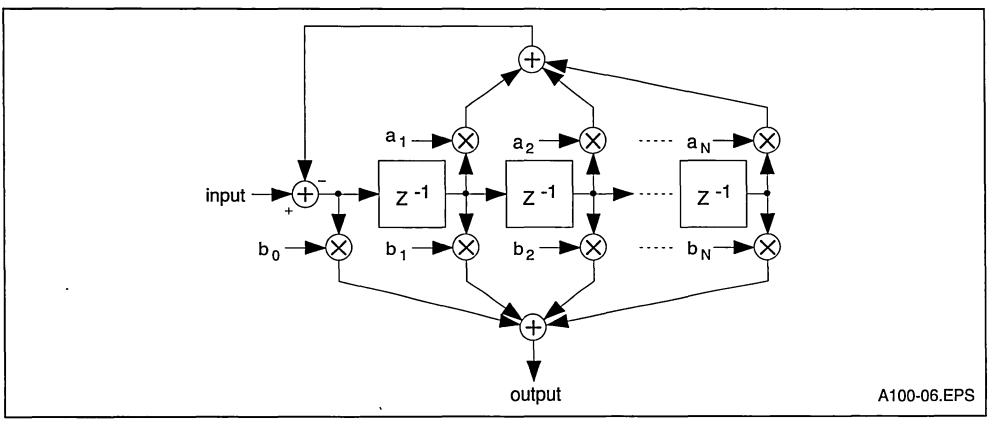


Figure 6 : Alternative Recursive (IIR) Digital Filter Structure with Reduced Number of Delay Stages



IV. DIGITAL FILTER DESIGN

Digital filter design methods can be divided into two categories:

(a) Design techniques suitable for FIR filters.
(b) Design techniques suitable for IIR filters.

In both cases the requirement is simply the choice of filter coefficients in such a way that the specification for the required transfer function is met. The IMSA100 can be used to implement high performance FIR filters directly. It can also be used to implement IIR filters, although the general problems associated with IIR filter design are then introduced. In this section a brief comparison between FIR and IIR filters is given and some of their associated design techniques are summarized. Where necessary the IMSA100 implementation issues are also discussed.

IV.1 Comparison between FIR and IIR filters

FIR filters, because of their finite-impulse response have no counterparts among analogue filters and as such can implement transfer functions which cannot be realized in the analogue world. One such property is the excellent linear-phase characteristic which can easily be realized with FIR filters. Since a linear-phase response corresponds to only a fixed delay, attention can be focussed on approximating the desired magnitude response without concern for the phase. The design techniques for FIR filters are generally simpler than those for IIR filters, and as there are no feedback paths in an FIR filter, the stability of the filter is guaranteed. Also FIR filters have been employed, and algorithms have been developed, for adaptive processing while the use of IIR filters in these types of systems is not common.

IIR filters on the other hand have infinite impulse responses and thus their design can be closely related to analogue filter design. IIR filters in general require fewer stages compared to FIR filters but their stability is not unconditional and great care should be taken to insure stability. Furthermore IIR filters do not generally result in linear-phase characteristics which is important in many applications.

IV.2 Basic design parameters

In digital filter design, for the reason of convenience, the frequency axis is usually normalised with respect to the sampling frequency f_s. For example for a filter with an actual pass-band cut-off frequency of 20kHz, a stop-band cut-off frequency of 30kHz and a sampling frequency of 100kHz we have:

$$\text{The normalised pass-band cut-off frequency } f_{pb}=\frac{20}{100}=0.2$$

$$\text{The normalised stop-band cut-off frequency } f_{sb}=\frac{30}{100}=0.3$$

As shown in Figure 7 the useful frequency axis (normalised) extends from 0.0 to 0.5, because the Nyquist sampling theorem requires a signal to be sampled at more than twice its highest frequency.

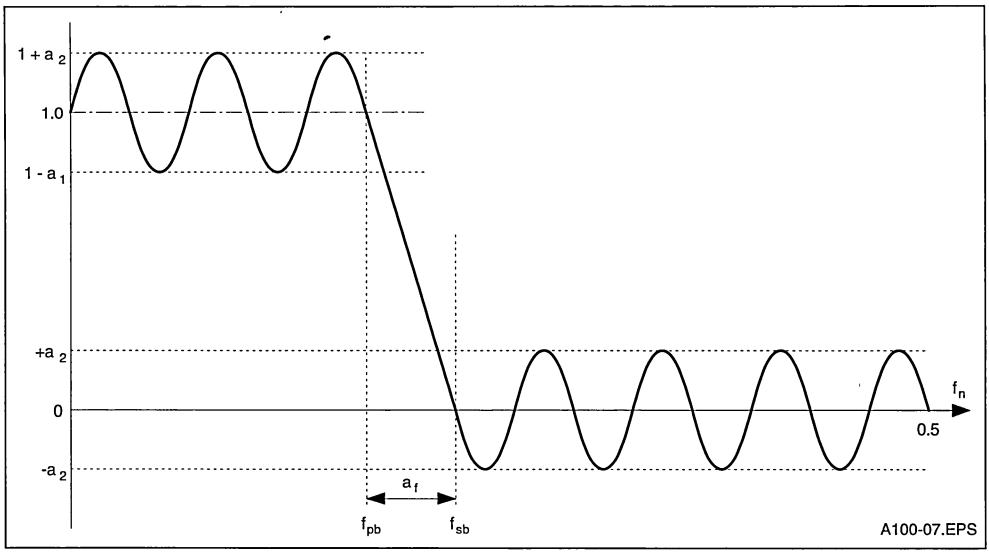
This means that the ratio of the frequency of any component in the signal to the sampling frequency must always be less than 0.5.

Referring to Figure 7, the pass-band and the stop-band ripples are usually expressed in dBs i.e:

$$\text{pass-band ripple (dB)} = 20 \log_{10}(1+\delta_1)$$
$$\text{stop-band ripple (dB)} = -20 \log_{10}(\delta_2).$$

The parameters f_{pb}, f_{sb}, δ_1, δ_2 and the sampling frequency define the basic specification of a filter prior to its design.

Figure 7 : Specification parameters for a low pass filter. Similar parameters exist for high pass and band pass filters



IV.3 Design techniques suitable for FIR filters

As mentioned earlier one of the major advantages of FIR filters is the ease with which linear-phase behaviour can be obtained from these types of filters. Before summarizing the design techniques for FIR filters let us briefly consider the necessary conditions for linear-phase behaviour. It can readily be shown that in order to obtain an FIR filter with a linear-phase characteristic, the following condition has to be met (references 1 & 2):

$$h(i) = \pm h(N-i) \quad \text{for } 0 \le i \le N \qquad (10)$$
$$= 0 \quad \textit{otherwise}$$

This condition requires that the the impulse response of the FIR filter, h(i), to have either positive or negative symmetry.

In the case of positive symmetry the frequency response will be of the form

$$H\left(e^{j\omega T}\right) = A(wT)\, e^{-j\omega TN/2} \qquad (11a)$$

where A(ω T) is a real function of ω. Notice that the phase is a linear function of frequency. These types of filters are appropriate for frequency selective filters.

In the case of negative symmetry the filter transfer function will have the following form:

$$H\left(e^{j\omega T}\right) = jB(wT)\, e^{-j\omega TN/2} \qquad (11b)$$

Again B(ω T) is a real function of ω. Note that the phase is again linear with frequency, but we also have a j term which indicates an extra phase shift of $\frac{\pi}{2}$. These types of frequency responses are required to realise approximate differentiators and Hilbert transforms which implement a $\frac{\pi}{2}$ phase shift over a specified frequency range.

There are essentially three well-established classes of design methods for (linear phase) FIR filters which are:

(i) window method
(ii) frequency sampling
(iii) timal design (Remez Exchange Algorithm)

Each one of these techniques has its own merits and the choice of which would depend on the application requirements and the design time involved.

IV.3.a. WINDOW METHOD

This is the most straight-forward approach to the design of FIR filters. In this method having defined an ideal frequency-response function, the corresponding ideal impulse response is determined by evaluating the inverse Fourier transform of the ideal frequency response. In the selection of the ideal frequency response, the linear phase condition may or may not be applied depending on the application.

As mentioned earlier because digital filters deal with signals sampled at a frequency f_s,it therefore follows that this frequency response is periodic in frequency with a period equal to f_s (Nyquist theorm). It is therefore possible to relate the impulse response and the frequency response of a digital filter via the following Fourier pairs:

$$H(\omega) = \sum_{n=-\infty}^{+\infty} h(n)\ e^{-jn\omega T} \qquad (12)$$

$$h(n) = \frac{1}{\omega_s} \int_{\frac{-\omega_s}{2}}^{\frac{+\omega_s}{2}} H(\omega)\ e^{jnwT}\, d\omega \qquad (13)$$

where ω_s, is the sampling frequency in radians/s and T is the sampling period. Having defined an ideal frequency response, H(ω), equation (13) can be used to obtain the impulse response, h(n), of the filter. As an example consider the ideal low-pass frequency response characteristics with a cut-off frequency ω_c as shown in Figure 8a. Using equation (13), and equating H(ω) to 1.0 for $-\omega_c \leq \omega \leq +\omega_c$ and to zero elsewhere, we can calculate the impulse response h(n) which is given by:

$$h(n) = \frac{\omega_c T}{\pi} \frac{\sin(n\,\omega_c\,T)}{(n\,\omega_c\,T)} \qquad (14)$$

where $-\infty < n < +\infty$. This impulse response is shown in Figure 8b. There are two problems associated with this impulse response obtained in this way:

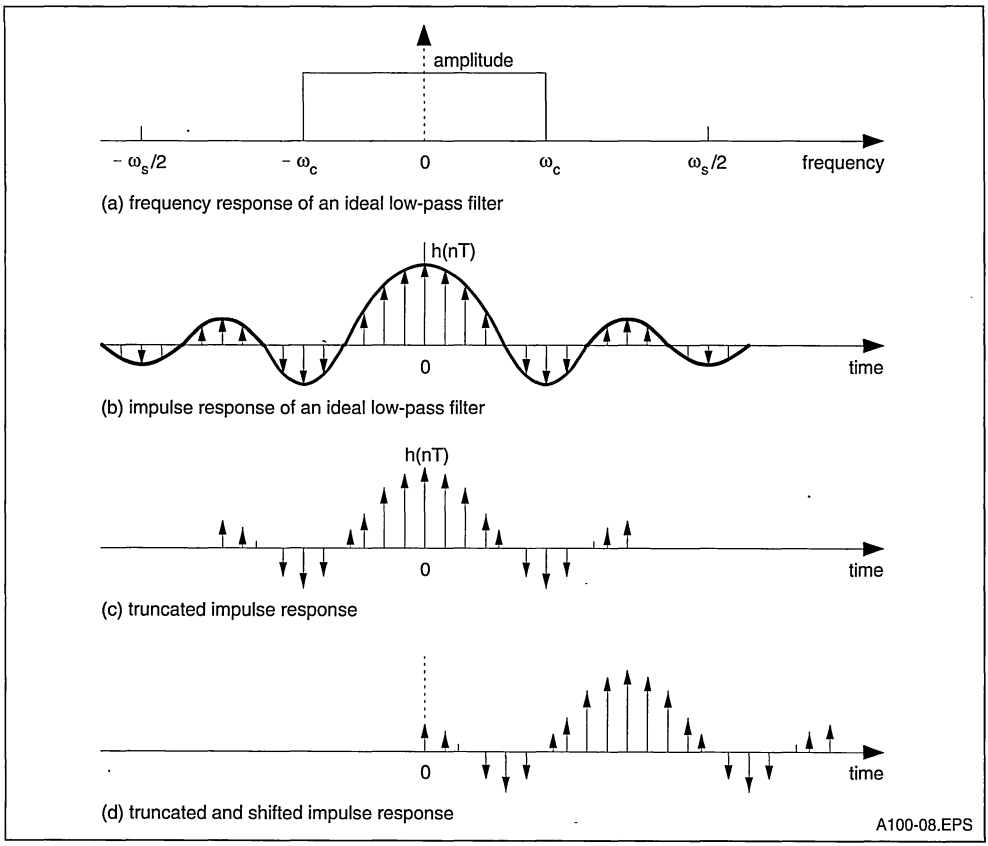
(i) The filter impulse response is infinite in duration and as such an FIR filter of infinite length is required (remember as discussed earlier for FIR filter the impulse response sample values are effectively the filter coefficients).

(ii) The filter is unrealizable since the impulse response begins at $-\infty$, indicating that no finite amount of delay can make the impulse response realizable.

One way to obtain an FIR filter which approximates the required frequency response is to truncate the infinite impulse response at $n = \pm\frac{N}{2}$, (see Figure 8c), and shift the impulse response to the right to avoid negative time (Figure 8d). This would result in a realizable FIR filter with N+1 coefficients which are equal to the impulse response samples.

The problem with this direct truncation of the impulse response is that it results in a fixed amount of overshoot (approximately 9%) before and after the discontinuity in the frequency response. In the literature this problem is referred to as the Gibbs phenomenon. For this reason, direct truncation is not often a reasonable way of designing FIR filters.

Figure 8



The frequency response of a truncated time series can be improved considerably by using a window function, w(n), which modifies the impulse response to w(n) × h(n). In the previous example the window was simply a rectangular window. Figure 9 shows the application of a different window function to the example of the ideal low-pass filter. Figure 9a shows the ideal infinite duration impulse response. Figure 9b shows the window function and Figure 9c shows the impulse response after the application of the window function. Figure 9d shows the shifted impulse response which avoids unrealizable negative delays. The filter coefficients (b_k's) correspond to the sample values of this modified impulse response which is now finite and realizable. Several window functions have been suggested in the literature some of which are:

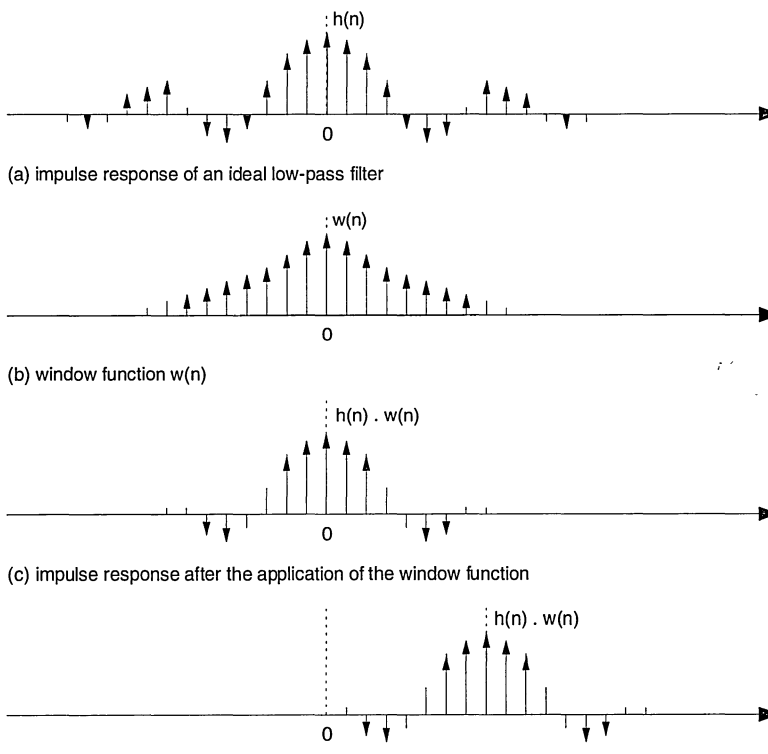
(i) Hamming window
(ii) Hanning window
(iii) Kaiser window
(iv) Dolph-Chebyshev window
(v) Blackman window

The generalized Hamming window function is given by:

$$w_H(n) = \alpha + (1-\alpha)\cos\left(\frac{2\pi n}{N}\right) \qquad (15)$$
$$= 0 \text{ otherwise}$$
$$\text{for } -\left(\frac{N-1}{2}\right) \le n \le -\left(\frac{N-1}{2}\right)$$

where $0 \le \alpha \le 1$. If $\alpha = 0.54$ the window is called a Hamming window, and if $\alpha = 0.50$ it is called a Hanning window.

Figure 9



For the Hamming window the main lobe of the frequency response is twice the width of that of the simple rectangular window. The amplitudes of the ripples of the Hamming window frequency response are considerably smaller than those of the rectangular window. For the rectangular window the peak side lobe (in the stop band) is only 14dB below the main-lobe (pass-band) peak. For the Hamming window the peak side lobe ripple is about 40dB below the pass band peak. Furthermore for the Hamming window 99.96% of the spectral energy is in the main-lobe peak.

Another family of windows are those proposed by Kaiser:

$$W_K(n) = \frac{I_0\left(\beta\sqrt{1-[2n/(N-1)]^2}\right)}{I_0(\beta)} \qquad (16)$$

$$= 0 \text{ otherwise}$$

$$for\ -\left(\frac{N-1}{2}\right) \le n \le -\left(\frac{N-1}{2}\right)$$

Where I_0 is the modified Bessel function of the first kind. The parameters β is used to specify the main-lobe width and the side-lobe level of the frequency response. β is usually specified to have a value between 4 and 9. This range of β corresponds to a range of side-lobe peaks of 3.1% to 0.047% of the main-lobe peak. The Kaiser window is essentially an optimum window in the sense that it is a finite duration sequence that has the minimum spectral energy beyond some specified frequency. For the Kaiser window the width of main lobe is almost three times that of the rectangular window, while the peak side lobe in the stop band is 57dB below the pass-band peak. The side-lobe ripple envelope decays to 94dB below the pass-band peak at half the sampling frequency.

The Dolph-Chebyshev window function has the minimum width of the main lobe in its frequency response for a given peak value of side-lobe ripple.

For this window the stop-band ripples all have the same amptitude. Recursive equations exist which allow this window function to be evaluated.

References 1 and 2 contain further information on this design method and the associated window functions.

IV.3.b. FREQUENCY SAMPLING TECHNIQUE

This technique is less common than the other two design methods, however for the sake of completeness it is briefly mentioned here.

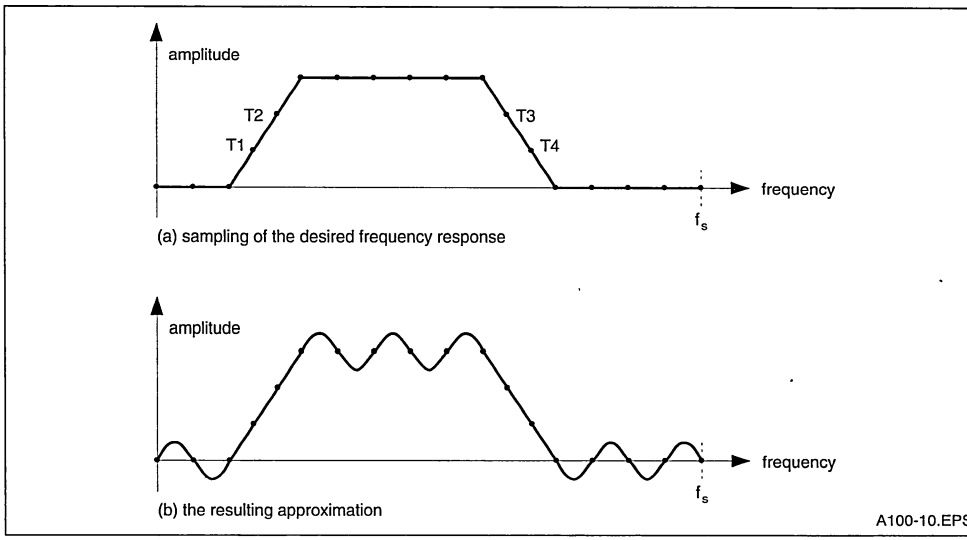
The basic idea behind this technique is that the given (desired) frequency response is approximated by sampling it at N equally-spaced points along the frequency axis between 0 and f_s (corresponding to N samples on the unit circle in the z-plane). An N-point inverse DFT is then performed on these N frequency samples to give N samples of the impulse response h(n) which corresponds to the filter coefficients. The z-transform of the filter impulse response is then given by

$$H(z) = \sum_{k=0}^{N-1} h(n)\ z^{-n}$$

Substituting $e^{j\omega T}$ for z, the resulting frequency response of the filter may be evaluated which would be an approximation of the desired frequency response. The approximation error would be exactly zero at points where the desired frequency response was sampled and would be finite between them. This process is depicted in Figure 10.

To reduce these approximation errors a number of frequency samples (particularly those in the transition band between band-pass and band-stop regions, i.e. points T_1, T_2, T_3 and T_4 in Figure 10 can be made unconstrained variables. The values of these unconstrained variables are then optimised using computer optimisation techniques involving linear-programming methods. This involves the solution of a set of linear unequalities in the unconstrained frequency samples. In this way, by adjusting the frequency sample values at T_1, T_2, T_3 and T_4, considerable ripple cancellation, both in the pass-band and stop-band, can be achieved resulting in very good filter characteristics. The detail of these techniques are beyond the objectives of this application note, however interested readers can refer to reference 1 for further information.

Figure 10



IV.3.c. OPTIMAL FILTER DESIGN – (REMEZ EXCHANGE ALGORITHM)

In the frequency sampling technique, discussed in the previous section, some degree of improvement in the filter characteristics is obtained by allowing only a few of the frequency samples to be adjusted via a linear-programming technique.

An even more powerful technique which results in truly optimal filters, in the sense of having the sharpest transition between pass bands and stop bands (for a given filter length and a given approximation error) has been formulated based on the so-called Chebyshev approximations. Computer optimisation techniques based on linear programming have been developed (references 3, 4, 5 & 6) which allowed engineers to design optimal FIR filters with a minimum amount of knowledge about the actual optimisation algorithm. These iterative algorithms are based upon the principles of the Remez exchange algorithm. This algorithm yields optimal filters that satisfy the so-called minimax error criterion (reference 1), where for a given number of coefficients, the filter minimizes the maximum ripple amplitude in the pass band. The implications of this optimal design are:

(a) The Remez exchange algorithm results in an FIR filter with the smallest number of coefficients satisfying the required specification.

(b) The pass-band ripple components all have the same magnitude and need not be equal to the stop-band ripples, but their ratio must be specified.

The input to the Remez exchange program usually includes the type of filter (frequency selective filters, differentiators and Hilbert transform filters), normalised stop-band and pass-band edges, the desired minimum stop-band attenuations, the maximum pass-band ripple and the ratio of the pass-band to stop-band ripples.

The output of the program include estimated filter length, and impulse response (filter coefficients). It also includes first pass computed values for design parameters, such as pass-band ripple, stop-band attenuation. If the computed values do not satisfy the design requirements, the filter length may be increased slightly and the program is run again. Interested readers can find copies of this program in references 1, 2 & 4.

IV.3.d. IMPLEMENTING FIR FILTERS WITH THE IMSA100

The coefficient word size in the IMSA100 can be programmed to be 4, 8, 12 or 16 bits. Having calculated the filter coefficients using one of the techniques described earlier, these coefficients are then expressed in a 4, 8, 12 or 16-bit format, depending on the required accuracy. The filter can then be implemented by simply loading these coefficients into the IMSA100 coefficient memories. If the number of coefficients (filter stages) required is less than or equal to 32, a single IMSA100 would be sufficient, any unused coefficient locations being set to zero. If however, more than 32 coefficients are involved a number of IMSA100 devices can be cascaded to obtain the required filter order. Alternatively it is possible to partition a long FIR transfer function into product terms where each term has an order equal or less than 32. Then, using a single IMSA100, the data can be recirculated through the same device with different coefficients (associated with each term in the transfer function) for each circulation. In this way a very long FIR filter can be implemented with a single device at the expense of a reduction in the data rate.

The IMSA100 can be cascaded very easily, without the need for any external components, to obtain high order filters with a high degree of accuracy. The device has a versatile architecture which allows it to be used in various system configurations. The coefficients can be programmed via a standard memory interface, while the input and output data can be communicated either via the memory interface or dedicated I/O ports. Figure 11 shows some of the possible system configurations for the IMSA100. In this diagram the interface between the host and the IMSA100 consists of data and address buses of the processor plus standard memory-type control signals such as R/W, CE and CS.

In Figure 11a the host processor controls the filter coefficients, while the actual data to be processed is supplied directly from an A/D to IMSA100. In this example the filtered output is fed directly to a D/A. Using the IMSA100 and a host processor it is possible to supply the input data to the device and also to collect the filtered samples via the memory interface. This allows system configuration such as those shown in Figures 11b&c. In Figure 11b the host processor receives the input data from a peripheral such as an A/D and writes it (may be after some preprocessing) into the data-input register (DIR) of the IMSA100. The filtered output sample is also collected by the host via the memory interface and output (possibly after post processing) to a peripheral such as a D/A. Figure 11c shows a configuration where the IMSA100 is used purely as

a signal processing accelerator to the host. Numerous other configurations are possible including integrating an IMSA100 into existing microprogrammed systems in order to improve the overall system performance.

As mentioned earlier large numbers of the IMSA100 devices can be cascaded to construct FIR filters of a high order. The cascading does not involve any external components and is simply a matter of connecting the output of the previous device to the cascade input of the next chip and joining the data input ports together (if they are being used rather than the memory interface). In normal operation the cascade input of the first device should be grounded. Figure 12a shows this cascading arrangement for two IMSA100 devices and Figure 12b depicts the block diagram of a system consisting of a host processor and two cascaded devices. In the latter case the data-input register (DIR) of both devices should be associated with the same address in the host's address space; and one of the devices should be selected as a master to generate the GO signal (see product data sheet for further detail).

Another important feature of the IMSA100 is a selector that is incorporated after the multiply-accumulator array. As discussed in the data sheet, the 32 multiply and accumulation in the array are performed to a precision of 36 bits which ensures that no intermediate overflows occur. The output selector can then be used to select and round a 24-bit word from this 36 bit result. This selection and rounding can be programmed to start from bits 7, 11, 15 or 20 and the selected word is sign extended if needed. One particularly useful selection is available when the input data and coefficients are in the form of 16 bit two's complement numbers normalised to between +1 and -1. In this case, if the selection is taken to start from bit 15, the output will have the same format as the input data (i.e. normalised to between +1 and -1).

Figure 11 : Possible System Configuration using the IMSA100 in Digital Filtering Applications

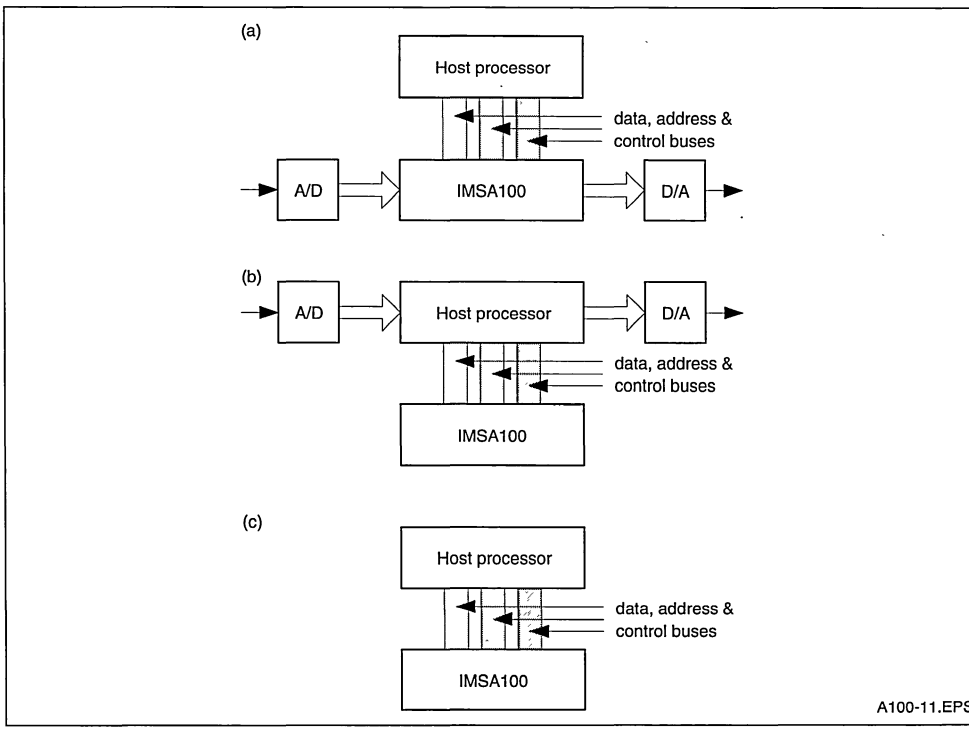
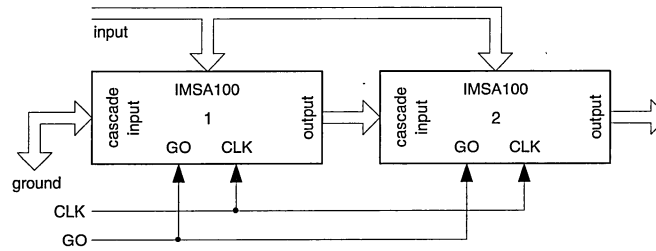
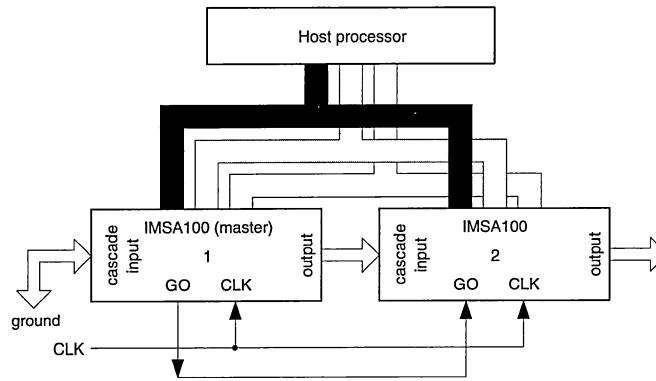


Figure 12 : Cascading IMSA100 devices



(a) cascading of IMSA100 devices using a dedicated input port



(b) cascading of IMSA100 devices when a host processor is used

A100-12.EPS

IV.4. The IMSA100 and IIR filters

Although the IMSA100 is designed primarily for FIR type filter implementations, it can also be used in realizing IIR filters. Referring to Figure 5 it can be seen that two IMSA100 devices can be used to implement an IIR filter of order 32 or less in the direct form. One chip performing the calculation in the feed-forward path while the other does the feed-back path. Note that in Figure 5 the output of the feed-back filter has to be combined with the input sequence in a subtractor and fed into the input of the second chip. This subtraction can be performed either by the host processor controlling the two IMSA100s or by an external adder.

A simpler and more elegant technique to implement IIR filters using IMSA100 is to make use of the continuous bank swap feature on the IMSA100 coefficient memories. This allows a single IMSA100 to be sufficient for the implementation of IIR filters whose order is less than or equal to 16. (Before describing how this can be achieved it is worth noting that IIR filters generally require considerably fewer stages than their FIR counterparts, and as such a 16th order IIR filter implementable on a single IMSA100 can be considered as having quite a high order). Figure 13 shows the coefficient memory allocations in this approach, where a's and b's are the feedback and feedforward coefficients of the IIR filter respectively (see Figures 5 & 6) and are loaded by the host processor. Note that in Figure 13 alternate coefficients are set to zero in the two memory banks. The chip is also set to the continuous bank swap mode so that in one cycle the feedback coefficients (a's) and in the next cycle the feedforward coefficients (b's) are used in the calculation. It will be shown in the following paragraphs that if the difference between data samples and alternate output samples are written to the data input register of the IMSA100, then the remaining output samples would correspond to the correct

filter output. The sequence of operations is as follows:

The host starts the filter operation by writing the first data value, x_0, to the data input register of the IMSA100. Remembering that the coefficient allocation is as shown in Figure 13, the first output of the device would be a_1x_0. Referring to Figure 6, it can readily be seen that this is indeed the feed back contribution needed to be subtracted from the next data sample x_1. The host reads this value (a_1x_0) from the data output registers (DOH and/or DOL) and stores it and then writes x_0, for a second time, to the IMSA100 input. This time the coefficient memory banks would have been swapped and the output would correspond to b_0x_0 which can readily be confirmed to be the first correct filter output (see Figure 6). The host then reads this result as the first valid sample of the filtered output.

Next the host subtracts the feedback factor, read in earlier (a_0x_0), from the second data sample x_1, and writes the difference to the input register of the IMSA100. Remembering that the memory banks are automatically swapped every cycle, the corresponding ouput of the IMSA100 will be:

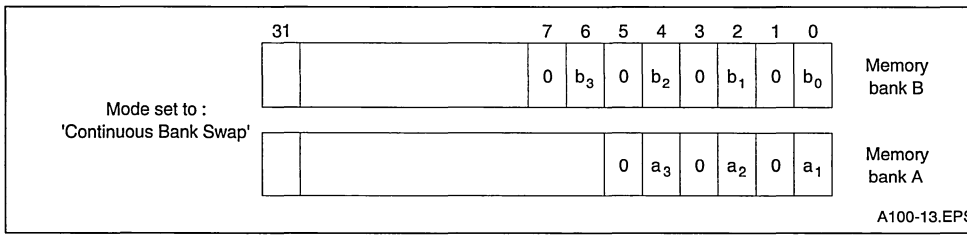
$$a_2x_0 + a_1(x_1 - a_1x_0)$$

Referring to Figure 6 you should be able to confirm that this value corresponds to the feedback contribution needed for the third input sample. The host reads this value and stores it and as before writes the input value (x_1-a_1x_0) to the IMSA100 input register for a second time. This will yield the second valid filtered sample i.e:

$$b_1x_0 + b_0(x_1 - a_1x_0) \quad (17)$$

The process is then continued in the same manner. The output of the IMSA100 will alternate between the feedback contribution and the filtered output samples. It should be emphasized that although the host is performing a single subtraction for every output value, it is the IMSA100 device which is performing the bulk of the processing. Having established how the IMSA100 can be configured to implement IIR filters, the next section deals with some of the design techniques that are used for determining the IIR filter coefficients.

Figure 13 : Coefficient Memory Allocation for IIR Filter Implementation



IV.5 Summary of the IIR filter design techniques

The problem of designing recursive filters is one of determining the feedforward and feedback coefficients (i.e. b_n's and a_m's in equation (8). The design techniques for IIR filters can be categorised into two basic groups:

(i) Indirect approaches.
(ii) Direct approaches.

IV.5.a. INDIRECT APPROACHES FOR THE DESIGN OF IIR FILTERS

As mentioned earlier digital recursive filters are closely related to conventional analogue filters. In the indirect method this similarity is exploited and the digital filter coefficients are determined from a suitable analogue filter, using some form of transformation technique. In other words the indirect approach uses the wealth of knowledge already available on analogue filters (such as Butterworth, Chebyshev and Elliptic filters) and develops a corresponding recursive digital filter. This method involves the following two steps:

(1) the determination of a suitable analogue filter transfer function H(s)
(2) transformation and digitization of this analogue filter

Some of the most popular design techniques falling into the indirect category are:

(a) Impulse-invariant transformation.
(b) Bilinear z-transform.
(c) Matched z-transform.

These three techniques can be employed to derive recursive digital filters from conventional analogue filter structures. Before discussing these three

techniques the basic characteristics of the common analogue filters, from which IIR filters are derived, will be briefly reviewed. The starting point in the indirect IIR design techniques is often one of the following analogue filter types.

1 Butterworth filters: These filters are characterised by the property that their magnitude characteristic is maximally flat at the origin of the s-plane. Butterworth filters are specified by their magnitude-square functions i.e:

$$|H(s)|^2 = \frac{1}{1+\left(\frac{s}{s_c}\right)^{2n}} \qquad (18)$$

The pole locations in the s-plane are equally spaced around a circle of radius $\omega_c (s_c = j\omega_c)$. These filters have a monotonically decreasing amplitude function with a roll-off of approximately 6ndB/decade. Figure 14 shows the overall amplitude response of this type of filter.

2 Chebyshev filters: In these types of filters the peak magnitude of the approximation error is minimized over a prescribed band of frequencies and is also equiripple over the band. Chebyshev filters are specified by the magnitude-square function:

$$|H(s)|^2 = \frac{1}{1+\varepsilon^2 C_N^2\left(\frac{s}{s_c}\right)} \qquad (19)$$

where $C_{N(s)}$ is a Chebyshev polynomial of order N. The parameter ε is used to specify a magnitude function with equal ripple in the pass band and monotonic decay in the stop band. Figure 15 shows the magnitude-square transfer function for the Chebyshev filter (type I) where the amplitude of the ripple is given by:

$$\delta = 1 - \frac{1}{\sqrt{1+\varepsilon^2}} \qquad (20)$$

The poles of the Chebyshev filter lie on an ellipse determined from the parameters ε, N and s_c. Chebyshev filters of type II on the other hand have monotonic behaviour in the pass band (maximally flat around ω_0) and exhibit equiripple behaviour in the stop band. For further details refer to references 1 & 2.

3 Elliptic filters:These filters exhibit a magnitude response that is equiripple in both the pass band and the stop band. These filters are optimum in the sense that for a given order and for a given ripple specification the transition band is the shortest possible. Elliptic filters are specified by the magnitude-square transfer function:

$$|H(j\omega)|^2 = \frac{1}{1+\varepsilon^2 C_N^2(\omega)} \qquad (21)$$

Where $C_{N(\omega)}$ is a rational Chebyshev function involving elliptical functions. Figure 16 illustrates the magnitude-square response for an elliptic filter.

Figure 14 : Frequency Response of the Butterworth Filter

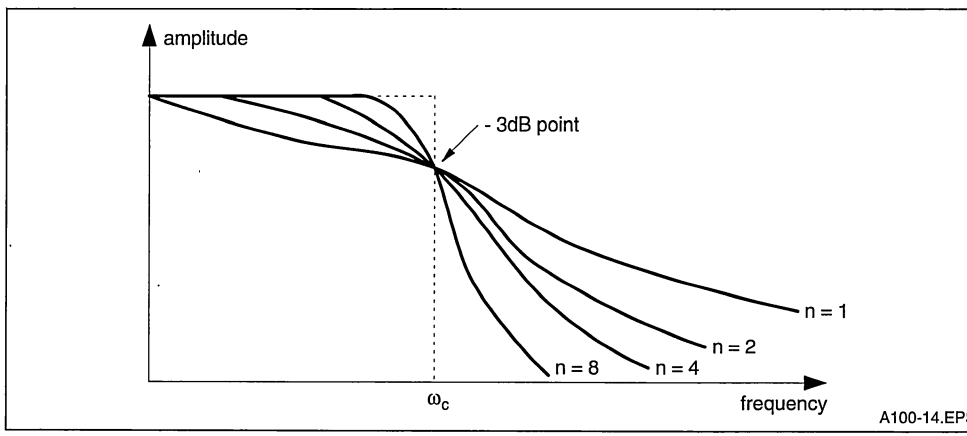


Figure 15 : Frequency Response of the Chebyshev Filter (type I)

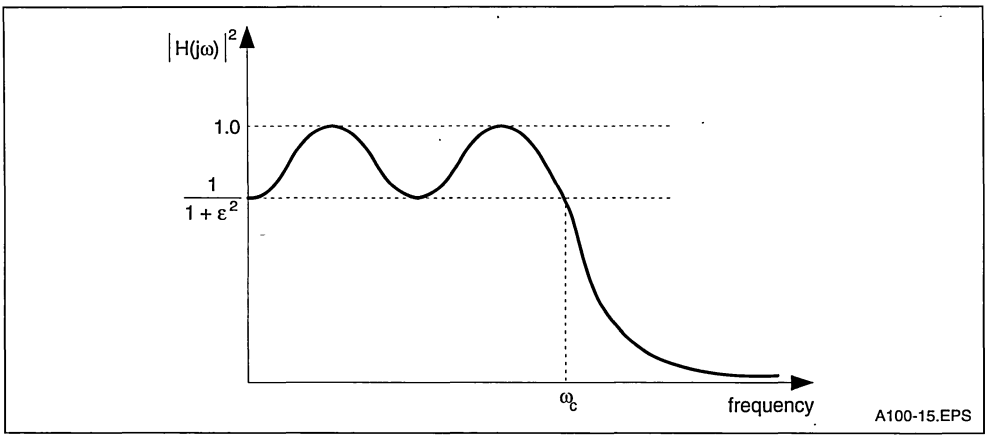
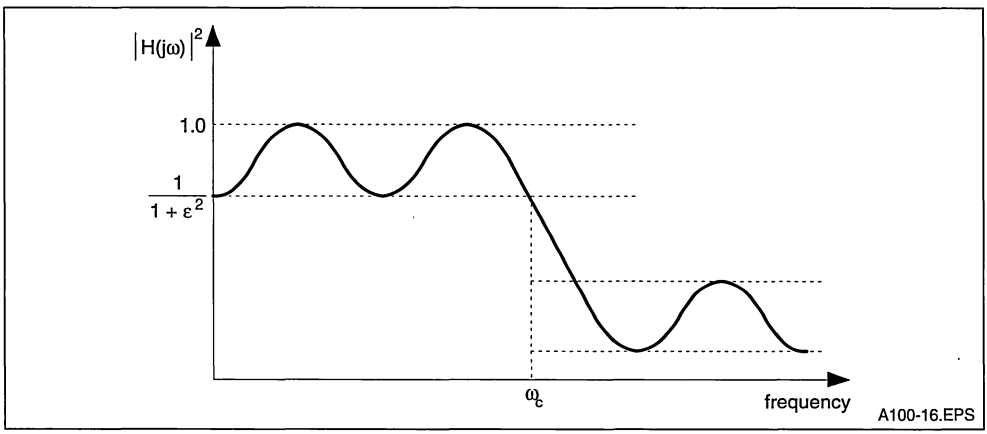


Figure 16 : Frequency Response for an Elliptic Filter



It is not possible to discuss all analogue filter types in this applications note as the main objective here is to summarize the basic design technique which allow transformation of analogue filters to digital realizations. Interested readers can refer to numerous books available on analogue filters.

Having decided the type and the specification of the analogue filter that satisfies the requirement, the next step in the indirect design method is to use one of the three following techniques to obtain the corresponding digital filter.

IV.5.b. IMPULSE INVARIANT TRANSFORMATION

One of the most common techniques for deriving a digital filter from a given analogue filter is the impulse-invariant transformation. As the name suggests this technique consists of using a sampled version of the impulse response of the analogue filter as the impulse response of the digital filter, i.e. the transformation does not change the impulse response of the analogue filter. Figure 17 illustrates the relationship between the analogue and the resulting digital responses of a typical low-pass filter obtained via the impulse-invariant method. The important point to note here is that sampling the analogue impulse response results in the frequency response of the resulting digital filter being periodic with a period equal to the sampling frequency f_s. This means that the digital filter will have

a frequency response similar to a repetitive version of that of the analogue filter. If the frequency response of the analogue filter does not decay to near zero beyond $\frac{f_s}{2}$ then serious aliasing would occur and the digital filter response would be corrupted. This aliasing problem means that this design technique is not suitable for high pass filters. However for low-pass and band-pass filters the problem can be avoided by choosing the sampling frequency high enough to ensure that the magnitude of the analogue filter response is negligible beyond $\frac{f_s}{2}$. (Note that the IMSA100 is capable of a sampling rate of 2.5MHz for 16-bit data and coefficients).

To demonstrate how the impulse-invariant transformation is used to digitize an analogue filter, consider the simple case of an analogue filter with an impulse response $h_{a(t)}=Ae^{-\alpha t}$ i.e. a simple RC filter (the s-domain transfer function of this filter is $\frac{A}{s+a}$). We start by sampling the impulse response of this analogue filter with a sampling interval T to obtain the corresponding impulse response for the digital filter, i.e.

$$h_a(kT) = A\,e^{-\alpha kT} \tag{22}$$

The z-transform of equation (22) is

$$H_d(z) = \sum_{k=0}^{\infty} A\,e^{-\alpha kT} z^{-k} \tag{23}$$

Noting that as equation (23) is a geometric series the result of the summation would be

$$H_d(z) = \frac{A}{1 - z^{-1}\,e^{-\alpha T}} \tag{24}$$

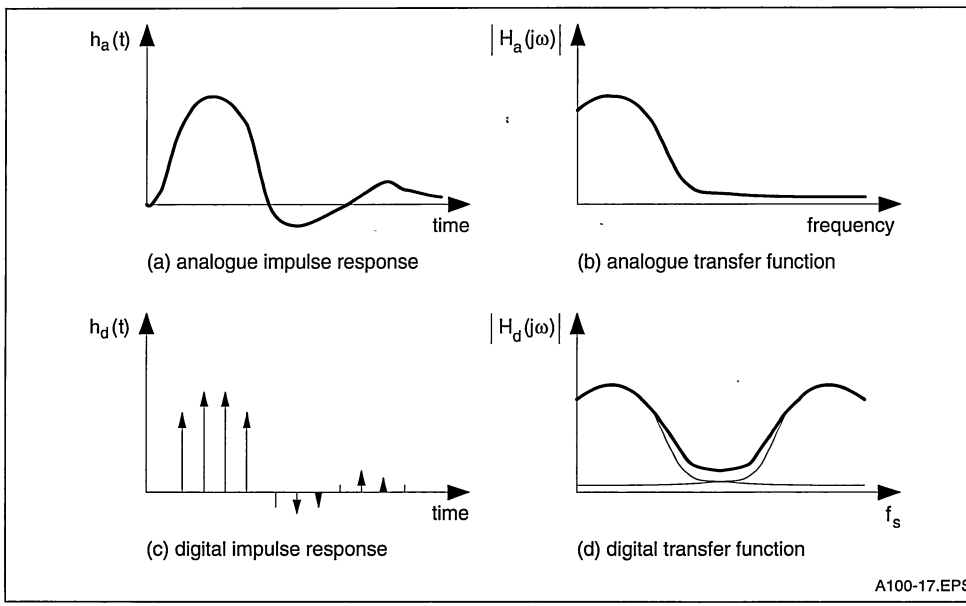
Equation (24) provides the z-domain transfer function of the resulting digital filter. To determine the filter coefficient (b_k's and a_m's), equation (24) can be compared with equation (8). For this simple example it can be seen that we have

$$a_1 = -\,e^{-\alpha T} \quad \text{and} \quad b_0 = A.$$

In this example, for the sake of clarity, the impulse responses were used to arrive at the z-domain transfer function. As analogue filters are often specified in the s-domain, it is more convenient to perform the impulse-invariant transformation directly from the s-domain to the z-domain. It should be obvious to the reader from the previous example that the required mapping is of the form

$$\frac{1}{s+\alpha} \Rightarrow \frac{1}{1 - e^{-kT} z^{-1}} \tag{25}$$

Figure 17 : The Impulsive Invariance Transformation Relationship between Analogue and Digital Impulse and Frequency Responses



It can be shown that this is indeed a general mapping (reference 1), applicable to the impulse-invariant method for both real and complex s-plane poles.

As a second example consider the two-pole analogue filter specified by:

$$H_a(s) = \frac{2}{(s+3)(s+1)}$$

expanding using partial fraction yields

$$H_a(s) = \frac{1}{s+1} - \frac{1}{s+3} \qquad (26)$$

Using equation (25) the digital transfer function would be:

$$H_d(z) = \frac{1}{1-e^{-T}z^{-1}} - \frac{1}{1-e^{-3T}z^{-1}} = \frac{\left(e^{-T}-e^{-3T}\right)z^{-1}}{1-\left(e^{-T}+e^{-3T}\right)z^{-1}+e^{-4T}z^{-2}} \qquad (27)$$

Again by comparing equation (27) with (8) we obtain the filter coefficients, $b_0 = 0$ $b_1 = e^{-T} - e^{-3T}$ and $a_1 = -(e^{-T}+e^{-3T})$ $a_2 = e^{-4T}$

As described earlier the sampling period T is chosen to ensure negligible aliasing in the filter transfer function.

IV.5.c. THE BILINEAR Z-TRANSFORMATION

Another indirect design method commonly used for recursive filters is the bilinear z-transformation. The major characteristic of this transformation is that it avoids the aliasing problem which was inherent in the impulse-invariant transformation. Given an analogue transfer function H(s), let us rename the variable s to s_a to indicate the reference to the analogue world i.e. $H(s) = H(s_a)$. Now let us define a new variable s_d related to s_a by the following mapping:

$$s_a = j\frac{2}{T}\tan\left(\frac{s_d T}{2j}\right) \qquad (28)$$

where T is the sampling period.

Since the analogue frequency variable ω_a is related to the s-plane variable by $s_a = j\omega_a$, we can also express the above mapping as:

$$\omega_a = \frac{2}{T}\tan\left(\frac{\omega_d T}{2}\right) \qquad (29)$$

where w_d is defined as $s_d = j\,\omega_d$.

Starting from an analogue transfer function $H(j\omega_a)$, Figure 18 illustrates the effect of this mapping on this transfer function. It can be seen from this diagram that the bilinear transformation compresses the entire analogue frequency range ($\omega_{a=0} \rightarrow \infty$) into a finite range equal to half the sampling frequency. This means that the spectral folding problem is completely eliminated and aliasing is therefore avoided. This compression of analogue frequency axis is usually referred to as frequency warping.

The price that is paid for this advantage is a distorted digital frequency scale resulting from this frequency warping. It can be seen from Figure 18 that due to the non-linear mapping the specification of the resulting filter, such as the cut-off frequency, would be somewhat different from the starting analogue filter. This distortion can be taken into account in the course of digital filter design. For example the cut-off frequency of the original analogue filters are modified slightly so as after the mapping the resulting filter has the desired cut-off frequencies.

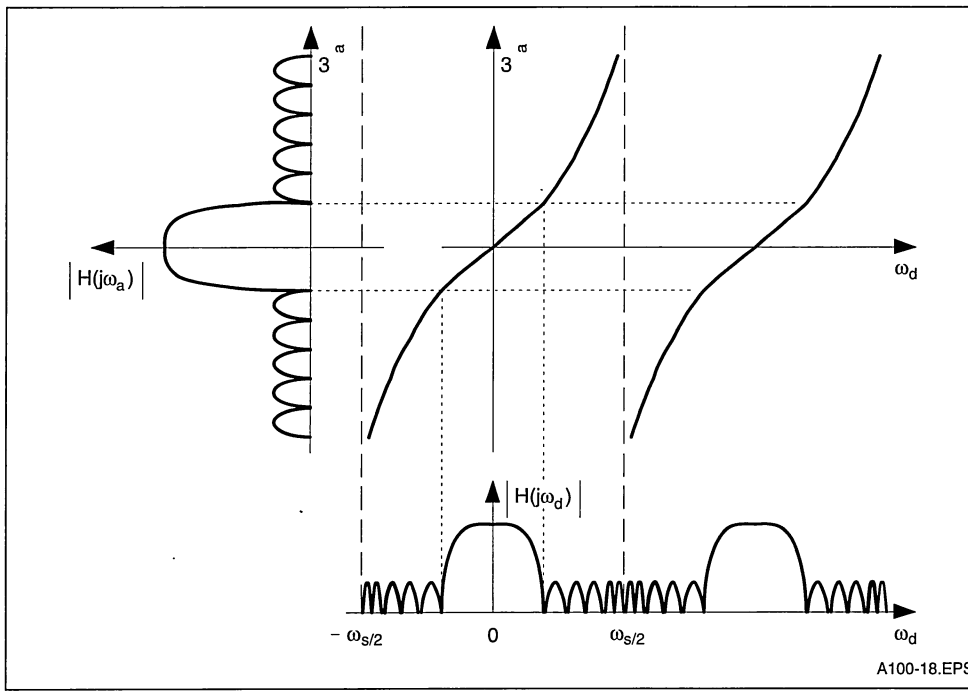
Returning to the transformation equation (28), we can rewrite it as:

$$s_a = \frac{2}{T}\left(\frac{1-e^{-s_a T}}{1+e^{-s_d T}}\right) \qquad (30)$$

and remembering that $z^{-1} = e^{-s_d T}$ we can write

$$s_a = \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \qquad (31)$$

Figure 18 : Graphical Illustration of the Bilinear-z-transform



Equation (31) provides the means for bilinear transformation directly from the s-domain to the z-domain suitable for digital filter implementation. To illustrate how the bilinear transformation technique is used consider the following example:

Filter specification

Low pass: 0 → 10kHz pass band

Sampling rate: 100kHz

Transition band: 10kHz to 20kHz

Stop-band attenuation: -10dB (starting at 20kHz)

Filter must be monotonic in pass and stop band.

Design

The monotonicity requirement indicates a Butterworth filter (see previous sections).

We have:

- digital filter cut-off frequency=$\omega_{cd}=2\pi \times 10000$
- start of digital filter stop band=$\omega_{sd}=2\pi \times 20000$.

Since the sampling rate is 100kHz, the sampling period would be $T=10^{-5}$

therefore $\omega_{cd}T=0.2\pi$ and $\omega_{sd}T=0.4\pi$

Using equation (29) we can calculate the corresponding analogue filter frequencies i.e.

analogue filter cut-off frequency $= w_{ca} = \frac{2}{T}\tan(0.1\pi)$
$= 0.6498 \times 10^5$

start of analogue filter stop band $= w_{ca} = \frac{2}{T}\tan(0.1\pi)$
$= 1.4531 \times 10^5$

The required order of the Butterworth filter can be determined by using equation(18) and ensuring at least 10dBs attenuation at $\omega = \omega_{sa} = 1.4531 \times 10^5$ i.e.

$$10 \log\left[1+\left(\frac{1.4531x10^5}{0.6498x10^5}\right)^{2n}\right] = 10$$

or

$$1+\left(\frac{1.4531x10^5}{0.6498x10^5}\right)^{2n} = 10$$

This gives n=1.367, therefore we choose n=2.

A second order butterworth filter with a cut-off at $\omega_{ca} = 0.650 \times 10^5$ has two equally-spaced poles on a circle of radius ω_{ca} (reference 1) given by

$s_1, s_2 = -0.6498 \times 10^5 (0.7071 \pm 0.7071j)$
$= -04595 (1.0 \pm j) \times 10^5$

and the transfer function is given by:

$$H(s) = \frac{s_1 s_2}{(s - s_1)(s - s_2)} = \frac{4.223 \cdot 10^9}{s^2 + 0.919 \cdot 10^5 + 4.223 \cdot 10^9}$$

Now we apply the bilinear-z transformation by substituting for s in the above transfer function from equation (31). This gives the following digital filter transfer function:

$$H(z) = \frac{0.0675 + 0.1349z^{-1} + 0.0675z^{-2}}{1 - 1.1430z^{-1} + 0.4128z^{-2}} \qquad (32)$$

The digital filter coefficients can be obtained by comparing equation (32) with (8) giving:

$b_0 = 0.0675$ ---
$b_1 = 0.1349$ $a_1 = -1.1430$
$b_2 = 0.0675$ $a_2 = 0.4128$

These coefficient values are then expressed in binary with the number of bits governed by the required accuracy. The factors affecting the necessary accuracy are discussed in section 5 of this application note.

Matched z-transform

This transformation is a direct mapping from the poles and zeroes in the s-plane to the poles and zeroes in the z-plane.

In general the two previous method i.e. the impulse invariant and the bilinear transformations are preferred to the matched z-transform as there are many cases where the matched z-transformation is not applicable. For this reason this technique is not detailed here. It would be sufficient to point out that the mapping is defined by the replacement relationship:

$$s + k = 1 - z^{-1} e^{-kT} \qquad (33)$$

IV.5.d. THE DIRECT DESIGN TECHNIQUES FOR IIR FILTERS

The IIR design techniques described so far were based on transforming a known analogue transfer function into the required digital filter transfer function. It is however possible to design digital IIR filters directly without reference to an analogue filter. Direct design methods fall into two categories namely direct closed form designs and optimisation techniques.

The direct closed form design techniques begin with the desired response of the filter from which one can often decide where to place poles and zeroes to approximate this response. These techniques are not very common and as such will not be discussed here.

The second classes of direct IIR filter design techniques are based on computer optimisation. In these approaches the set of design equations cannot be solved explicitly, instead mathematical optimization techniques are employed to determine the filter coefficients that minimize some error criterion, subject to a set of design equations. The algorithms involved in these optimisation techniques are of an iterative nature and are terminated when the error reaches a minimum or the number of iterations exceeds a specified limit.

Among the most commonly used optimisation technique is one which minimizes the pass-band ripples in filters exhibiting a given stop-band attenuation. This technique is sometimes referred to as the minimax method and the optimization algorithm involved has been developed by Fletcher and Powell (reference 7). The Fletcher-Powell optimization algorithm generates the filter coefficients by using a convergent descent method.

The spectral flatness approach is another optimisation technique and is based on the fact that multiplying the desired frequency response by its inverse should result in unity throughout the frequency spectrum (i.e. a flat spectral line). Any deviations from the ideal response would result in ripples in this flat spectral line. Optimisation techniques have been developed which attempt to minimize these ripples (reference 8). The difficulty with this technique is the modeling of the desired frequency response.

Mean-square-error optimization techniques have also been developed for IIR filter design. One such technique has been described by Steiglitz (reference 9) which involves minimizing the square of the difference between actual filter behaviour and the desired performance. This algorithm searches an error .vs. design-parameter curve for a local minimum.

The details of the above optimisation techniques are beyond the objectives of this application note. However the references given should prove adequate for interested readers.

V. FINITE WORD-LENGTH CONSIDERATIONS AND PROBLEMS

In implementing digital filters both the input samples and the filter coefficients have to be quantised and expressed in a limited number of bits. In the IMS A100 chip both the coefficients and data samples can be quantized up to 16-bits of accuracy, although smaller word-lengths can be used if desired.

The problems of finite word length in digital filters apply to both FIR and IIR filters but their implications are much more severe for the IIR filters, due to their inherent feedback nature. In the fixed-point implementations of digital filters it is usual to normalize the numbers so as to make their absolute values less than one i.e. in the form of

$$d_{N-1} . d_{N-2} d_{N-3} \text{ --------- } d_5 d_4 d_3 d_2 d_1 d_0$$

where d_n represents the nth bit in the word and (.) indicates the binary point. Using this format (and two's complement notation) the number

0.1111 ... 1111

would represent a value very nearly equal to +1, while the number

1.0000 ... 0000

would represent a value equal to -1.0.

If purely integer numbers were to be used the process of truncation or rounding after multiplications would become meaningless. However using the above fractional-number representation, where the numbers are normalized to be less than one, the problems would not arise as the product of two numbers which are less than one would also be less than one.

In general there are three sources of error arising in the implementation of digital filters these are:

(i) Finite precision of the filter coefficients
(ii) Limited word length of the input data
(iii) Round-off and truncation errors in the multiplication and addition operations.

The finite precision in the representation of the filter coefficients will obviously cause the frequency response of the filter to depart to some extent from that desired for both FIR and IIR filters.

Furthermore in the case of the recursive IIR implementation, because of the existence of feedback paths, this finite precision may cause instabilities in the filter behaviour. This happens because the inaccuracies may move the z-plane poles outside the unit circle hence causing instabilities. The chances of this happening depends on how close the poles are to the unit circle in the first place. If multiplication and addition operations are followed by truncation and rounding (in order to contain word growth) further difficulties may arise. These problems may manifest themselves in undesirable oscillations in the form of 'limit cycle' or 'overflow' oscillations (discussed later). It is therefore absolutely essential for the filter behaviour to be simulated using the precision and roundings involved in the intended implementation. This is particularly relevant to recursive IIR filter where a risk of instability exists.

One of the consequences of rounding and quantisation in the digital recursive(IIR) filters is the limit-cycle phenomenon, which takes the form of a stable periodic non-zero output for zero or constant input. The limit cycle behaviour of a digital filter in general is complex and difficult to analyse. However for simple first order filters, it is possible to illustrate the effect by way of an example. Consider the first order recursive filter with the following equation:

$$y(n) = 0.09 \times (n) + 0.91\ y(n - 1)$$

Assume that each output y(n) gets rounded to the nearest integer, also assume that the input is constant at 100 and the previous output is 90.

The following table shows the resulting rounded output sequence for each iteration.The last column shows the perfect output (without rounding) for comparison.

n	x(n)	y(n)	rounded y(n)	perfect y(n)
0	100	--	90	--
1	100	90.9	91	90.9
2	100	91.81	92	91.72
3	100	92.72	93	92.46
4	100	93.63	94	93.14
5	100	94.54	95	93.76
6	100	95.45	95	94.32
7	100	95.45	95	94.83
8	100	95.45	95	95.30
9	100	95.45	95	95.72
10	100	95.45	95	96.11
..	...	.....	..	.....
..	...	.....	..	.....
..	...	.....	..	.....
..	100	95.45	95	100.0

It is observed that the output sticks at a value of 95. However if the same filter is implemented with very high precision and no rounding the filter output would closely approach 100 (last column in the table).

If we approach the limit from the opposite side by starting with a value of y(n) of say 110, the output would arrive at a limit of 105. You can see from this example that the system has a dead zone of ± 5 units around the ideal output of 100.

In fact it can mathematically be shown that for a first-order recursive filter of the form

$$y(n) = b\,x(n) + a\,y(n-1)$$

The dead zone is given by

$$|\,dead\ zone\,| \leq \frac{\frac{q}{2}}{1-|a|} \qquad (34)$$

where q is the quantisation step. In the above example a quantised step of 1 was used and equation (34) gives a dead zone of ± 5 too.

For second-order systems similar results to (34) have been derived in the literature (reference 1).

Overflow oscillation is another problem associated with digital recursive filters. In the IMSA100 chip the full internal precision ensures that no overflow occurs in the multiply-accumulator array. The only source of possible overflow is the external addition which is performed in combining the feedback terms with the input samples (see section 4.4). A simple but effective way to eliminate these oscillations is to perform this addition in a saturating manner (similar to analogue adders). This operation can easily be taken care of by the controlling host processor.

In the IMSA100 device the data and coefficients can be expressed to a precision as high as 16 bits. The 32 multiplications and additions are carried out to 36-bit precision. This ensures that no overflow occurs in the multiply-accumulation array (unless all the coefficients and 32 consecutive data items have values equal to the most negative 16-bit number i.e. 1000000000000000 in binary, which is of course highly unlikely). The selector at the output of the multiply and accumulate array allows the rounding and selection of 24 bits out of this 36 bits. The combination of full internal accuracy, the selector functionality and the fact that the IMSA100 devices can easily be cascaded allows high quality FIR filters to be readily implemented. As described earlier the device can also be used to implement efficient IIR filters only in direct forms. It is well known that for high order filters direct implementations of IIR filters are more prone to instabilities compared to cascade or parallel arrangements. However the full internal precision of the IMSA100 combined with comprehensive filter simulations should minimize these instabilities. It should however be emphasized that it is possible to implement a high order high precision IIR filter in the cascade form on the IMSA100 at the expense of processing speed. In this case the IMSA100 should be used to implement low order (2nd or 4th order) sections of a cascade arrangement in turn by reloading suitable coefficients. The functionality of the whole filter is obtained by recirculating the first output batch through the chip with its coefficients modified to implement the 2nd section in the cascade array and so on.

For the IIR filter implementations Figures 11c & 11b can be considered as possible system configurations.

VI. ADAPTIVE FILTERS

So far we have discussed digital filters with fixed characteristics. Fixed filters are used in many practical situations to combat noise or interfering signals (e.g. a matched filter) or to select a desired frequency band (e.g. a band-pass filter). In digital signal processing the parameters of such fixed filters are determined once and remain unchanged during processing. Adaptive filters on the other hand automatically adjust their own parameters and seek to optimize their performance according to a specific criterion. The adaptive nature of such filters makes them particularly suitable for situations where signal properties are unknown or variable with time.

Figure 19 illustrates the basic structure of an adaptive filter. The input signal x(t) is filtered or weighted in a programmable filter to yield an output y(t). The filter output y(t) is then compared with a reference (sometimes called a training signal) waveform to yield an error signal e(t). This error is then used to update the filter coefficients in such a way that the error is progressively minimized. Several algorithms for updating the filter coefficients have been developed and can be found in references 10, 11 & 12.

One example of adaptive filtering is echo cancellation in telephony. Echoes are the result of impedance mismatches in the communication circuits. The hybrid couplers which are used at the interface between two-wire and four-wire circuits are a major source of echoes. Figure 20 shows how an adaptive filter arrangement can be used to cancel these echoes at the hybrid interface. Notice that in this case the training signal contains the echo, while the input to the adaptive filter is the signal arriving at the hybrid. Effectively the filter adaptively models the echo path and produces a synthetic antiphase echo return which cancels the echo in the 4-wire path returning from the hybrid.

Adaptive filters have application in low-bit rate speech coding based on linear prediction where the filter coefficients, after adaption, are transmitted instead of the speech signal itself.

The programmability of the IMSA100 can be exploited in the implementation of adaptive filters as well as fixed filters discussed earlier.

Figure 19 : Basic Structure of an Adaptive Filter

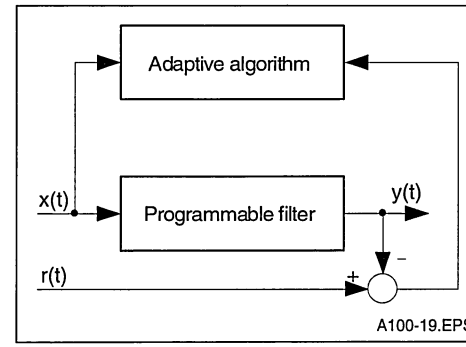
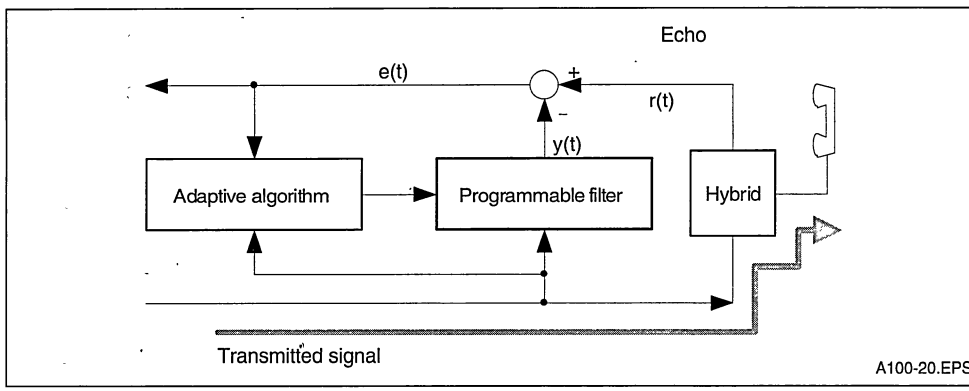


Figure 20 : Application of Adaptive Filtering Techniques to Echo Cancellation



VII. REFERENCES

1 Theory and application of digital signal processing, Rabiner L.R., and Gold B., Prentice-Hall Inc, Englewood Cliffs, NJ, I975.

2 Digital signal processing, Oppenheim A.V., and Schafer R.W., Prentice-Hall Inc, Englewood Cliffs, NJ, I975.

3 A computer program for designing optimum FIR linear-phase digital filters, McClellan J.H., Parks T.W., and Rabiner L.R., IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, No.8, December I973.

4 Digital signal processing, Peled A., and Liu B., John Wiley and Sons Inc., New York, I976.

5 Practical design rules for optimum finite impulse response low-pass digital filters, Rabiner L.R., Bell Systems Technical Journal, Vol. 52, No.6, July-August I973.

6 Approximate design relationships for low-pass FIR digital filters, Rabiner L.R., IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, No.5, October 1973.

7 A rapidly convergent descent method for minimization, Fletcher R., and Powell M., Computer Journal 6 (No.2) I963, pp 163-168.

8 Time series analysis: Forcasting and control, Box G., and Jenkins G., Holden Day, San Francisco, CA, I975

9 Computer aided design of recursive digital filters, Steiglitz K., IEEE Transactions on Audio and Electroaco ustics 18 (No.2), June I970, pp123-129.

10 Adaptive noise cancelling: principle and applications, Widrow B. et al, Proc. IEEE, I975, Vol.63, No.12, pp 1692-1716.

11 Design and application of adaptive filters, Grant P.M., Cowan C.F.N., Electronics & Power, February I985.

DISCRETE FOURIER TRANSFORM WITH THE IMSA100

I. INTRODUCTION

In the time-domain representation, signals are expressed as a function of time. For example $x = Ae^{-at}$ is a time-domain description of a signal whose amplitude decays exponentially with time.

In the 18th century J.B. Fourier showed any signal that can be generated in a laboratory can be expressed as a sum of sinusoids of various frequencies. In other words each signal can be said to have a frequency spectrum represented by the amplitude and phases of various sinosoidal components. The frequency spectrum of a signal completely specifies it and is referred to as frequency-domain description of the signal.

The Fourier integrals provide the means for obtaining the frequency-domain representation (the spectrum) of a signal from its time-domain representation or vise-versa, i.e.

Fourier Transform:

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt \qquad (1)$$

Inverse Fourier Transform:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}d\omega \qquad (2)$$

where x(t) is a time-domain signal, $X(\omega)$ is the frequency spectrum of x(t) and ω is the frequency variable.

These transforms are fundamental to the description of many real world phenomena in the fields of science and engineering. In the area of signal processing the Fourier transform is an important mathematical (and nowadays practical) tool in understanding, analysing and solving system-level problems. The Fourier transform allows us to translate time serial information into the frequency domain in a reversible way. The components of a signal, although dispersed in the time domain, may have restricted occupancy or a characteristic relationship in the frequency domain. In fact many physical processes can be categorised by the frequencies they generate and their relative strength. This is one reason why the Fourier transform, with

its ability to segregate frequency components of a signal, has gained such an importance in many signal processing environments.

Apart from the ability of the Fourier transform to provide spectral information, many signal processing functions such as correlation, filtering and beamforming can be expressed in terms of the Fourier transform and its inverse. For these reasons considerable effort has gone into the development of efficient algorithms for evaluating the Fourier transform. The continuous-time Fourier transform, given by equation (1), can be made suitable for digital computation by sampling the time and frequency variables and limiting the computation to a finite set of data points. This modified version of the Fourier Transform is often referred to as the Discrete Fourier Transform (DFT) and will be discussed in more detail in the next section.

Many algorithms have been developed for efficient digital computation of the DFT. Until recently the digital multipliers needed to implement DFT's were costly, large and relatively slow, and the general purpose microprocessors were extremely slow at performing multiplications. Consequently it was necessary to calculate DFT's using a minimum number of multiplications and to use data and coefficient storage economically and this led to development of several Fast Fourier Transform (FFT) (references 1 & 2) algorithms. These FFT algorithms make use of the redundancies, that occur in the DFT, to reduce the arithmetic operations involved. Most FFT algorithms were designed simply to minimise the total number of multiplications required to calculate the DFT, often at the expense of an increase in the number of additions, memory accesses and control complexity. One such algorithm is the Cooley-Tuckey radix-2 FFT algorithm which necessitates a data size equal to a power of two ($N=2^n$). Winograd FFT algorithms on the other hand requires that the number of data points to be prime. Both algorithms simply minimize the number of multiplications in the DFT by the use of redundancies resulting from the particular choice of the data size. These algorithms are particularly suitable for general-purpose computers and microprocessors where the major limit on processing speed is the time taken to perform the multiply instruction.

Other algorithms have been developed which map the DFT process into particular hardware structures. Two such techniques are the Rader's Prime Number Transform (PNT) (reference 3) and the Chirp-Z Transform (CZT) (reference 4) which convert the DFT into circular correlation/convolutions. These algorithms are particularly suitable for implementation using transversal filter type structures. In the past, CCD and SAW transversal filters have been used to implement high-throughput wide-bandwidth DFT processors using these algorithms. The analogue nature of the CCD and SAW technologies has restricted the precision of these processors.

The availability of the IMSA100, the first high performance cascadable digital transversal filter, means that the same algorithms can now be implemented digitally offering both high speed and high accuracy. This application note deals with the concepts behind these algorithms and their implementations using the IMSA100 signal processor. Generalised mapping techniques which facilitate the DFT evaluation of a long data sequence via a number of short transforms are also discussed (The radix-2 FFT is a special case of these more general partitioning techniques). The approach described here is particularly suitable if a long DFT is to be evaluated with a transversal filter of limited size. Also, these decomposition methods are applicable to concurrent architectures and as such provide the basis for the trade-off between speed and cost involved in a particular implementation. These issues are of major importance when combining the IMSA100(s) with the INMOS transputer family of parallel processors.

Figure 1a shows the structure for an N-stage canonical transversal filter where the output is the weighted sum of the N most recent input samples. The IMSA100 implementation of the transversal filter is depicted in Figure 1b where the multi-input summation of the canonical form has been replaced by a delay and add chain. You should be able to convince yourself that the two structures in Figure 1 have the same functional behaviour. The main difference is that in Figure 1b the partial product terms are passed down the delay-and-add chain whilst in Figure 1a, the input samples are delayed and the sum of products is calculated simultaneously.

A simplified functional diagram for the IMSA100 is shown in Figure 2. The major processing part of the chip incorporates 32 multipliers and a 32-stage delay-and-add chain. For the IMSA100 the input data word length in 16 bits. The coefficient word length can be programmed to be 4, 8, 12 or 16 bits. The data throughput ranges from 2.5 million

samples/s to 10 million samples/s depending on the coefficient word size. Two complete sets of coefficient memories are provided. At any instant one set of coefficients is applied to the transversal filter, whilst the other set can be accessed via a standard memory interface (capable of 100ns cycle time). The function of the two coefficient memories can be exchanged by writing to control registers. Further this exchange can be made continuous, i.e alternate sets of coefficients can automatically be selected for successive computation cycles. This is particularly useful for complex number processing.

Data input and output are available both through dedicated ports or via the memory interface. This selection can be programmed via the control and status registers in the IMSA100.

To preserve complete numerical accuracy, no truncation or rounding is performed on the partial products in the multiplications and delay-and-add chain. The output of the chain is calculated with a precision of 36 bits which is sufficient to ensure no overflow occurs (the only time that the output of the delay-and-add chains exceeds 36 bits is when all 32 coefficients and 32 successive input samples have the maximum possible negative value i.e. 1000000000000000 in two's complement binary notation, this is of course highly unlikely). A programmable barrel shifter is located at the output of this chain, which allows 24 bits (starting at bits 7, 11, 15 or 20 of the full 36 bit result) to be selected and rounded for output. To allow devices to be cascaded without any external components, a 32-stage 24-bit wide shift register and a 24-bit adder are included on the chip. For cascading purposes the output of one chip is connected directly to the cascade input of the next.

The control registers accessible via the memory interface allows various operational parameters to be programmed. For the full detail of the specification you are advised to refer to the IMSA100 data sheet.

In the following parts of this application note the basic concepts of DFT will be reviewed and some algorithms for its evaluation will be summarized. This is followed by a detailed description of those DFT algorithms suitable for implementation using the IMSA100 transversal filter. A multi-dimensional mapping technique is also described which allows efficient computations of long-length DFTs via the IMSA100 implementations.

Figure 1 : Transversal Filter Architecture

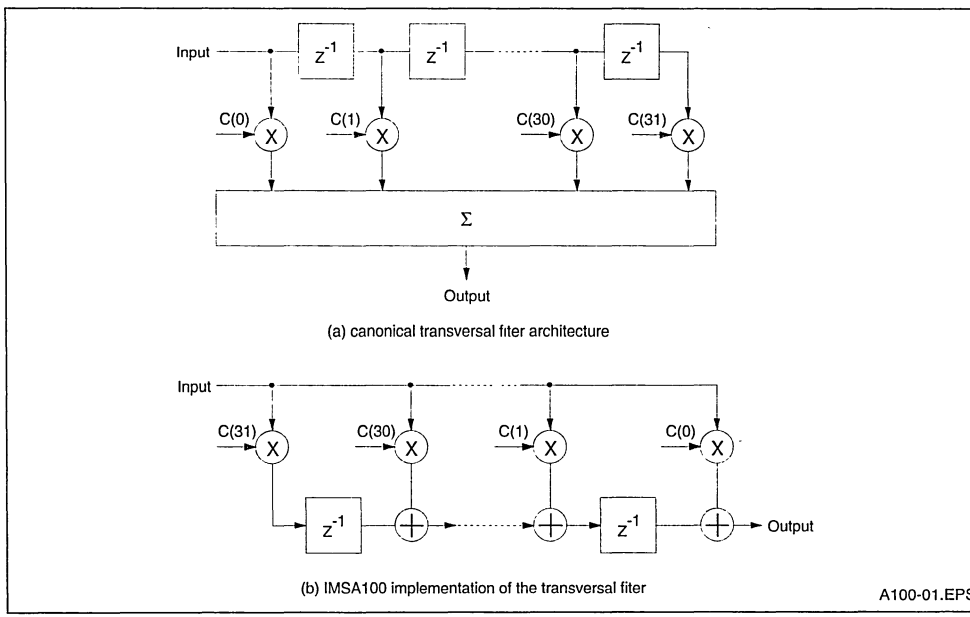
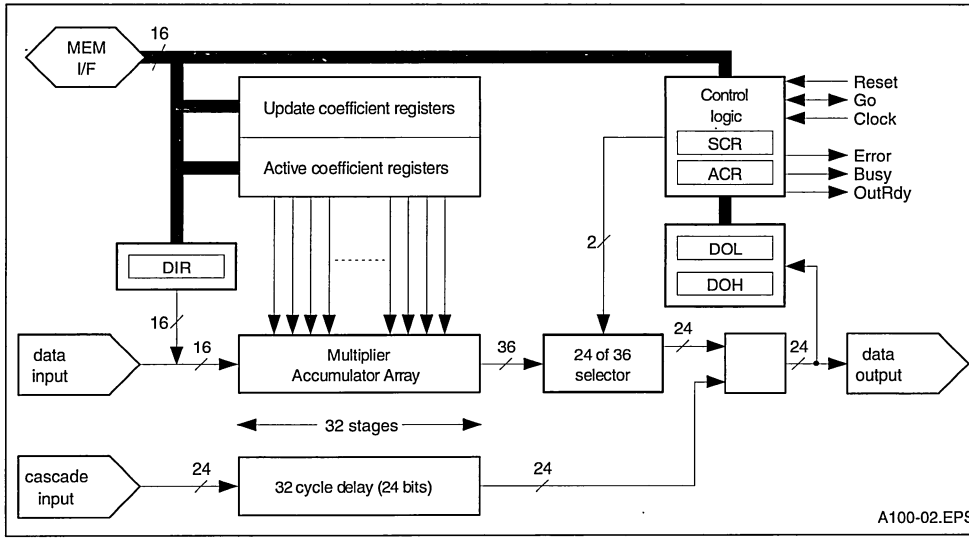


Figure 2 : User's Model of the IMSA100



II. THE BASIC CONCEPTS OF DFT

The equation for the Fourier transform and its inverse (equations 1 & 2) can be made suitable for digital processing by discretizing both the time variable t and the frequency variable ω; and by constraining the integration to finite limits. Referring to equation 1, for the forward transform, this can be done by making t=nT and ω=kω0. Where T is the sampling period of the time function and ω_0 is the frequency resolution of the discrete spectrum. If the integration limits are confined over N time samples for which N independent frequency samples can be calculated we have

n = 0,1,2,3, ...,N-1
k = 0,1,2,3, ...,N-1

The Fourier-transform integral then becomes a Fourier-transform sum given by:

$$X(k\omega_o) = \sum_{n=0}^{N-1} x(nT)\, e^{-jk\omega_o nT} \quad k = 0, 1, \ldots N-1 \quad (3)$$

It can be shown that the frequency resolution in the f-domain is given by:

$$\omega_o = \frac{2\pi}{NT} \quad (4)$$

Substituting this in (3) gives:

$$X(k\omega_o) = \sum_{n=0}^{N-1} x(nT)\, e^{-2\pi jkn/N} \quad k = 0, 1, \ldots N-1 \quad (5)$$

For convenience the terms T and ω_0 are usually dropped from the indicies giving the DFT equation as:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-2\pi jnk/N} = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \quad (6)$$
$$k = 0, 1, \ldots N-1$$

where

$$W_N = e^{-2\pi j/N} = \cos\left(\frac{2\pi}{N}\right) - j \sin\left(\frac{2\pi}{N}\right) \quad (7)$$

The inverse discrete Fourier transform (IDFT) can be derived in a similar manner from its corresponding continuous form and is given by:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, e^{2\pi jnk/N} = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, W_N^{-nk} \quad (8)$$
$$n = 0, 1, \ldots N-1$$

Note that the DFT and its inverse, IDFT, are very similar, the only difference is the factor $\frac{1}{N}$ and the negative exponent in the IDFT. This similarity has important practical significance as it allows an algorithm or a hardware developed for DFT to be used for IDFT with minor modifications. For example an inverse DFT on the data sequence x(0), x(1) ... x(N-1) can be carried out by first reversing this sequence to generate a new data set x'( ) such that x'(0)=x(N-1), x'(1)=x(N-2).....x'(N-1) = x(0) and then performing a DFT and dividing the result by N . This technique works simply because x'(k) = x(-k) , (In the DFT both x(n) and X(k)

are assumed to be periodic) which converts the positive exponential in (8) into a negative one, representing a DFT. For this reason any algorithm or implementation in the following subsections will only be described for DFT as the extension to an IDFT is trivial.

It is worth noting that some authorities write the DFT and its inverse as:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, e^{-2\pi jnk/N} \quad \text{DFT}$$

$$x(n) = \sum_{k=0}^{N-1} X(k)\, e^{2\pi jnk/N} \quad \text{IDFT}$$

i.e. the factor $\frac{1}{N}$ is applied to the DFT rather than its inverse. This version can be seen to have a physical meaning since X(0) , as defined, represents the average of the sampled time waveform i.e. the 'd.c.' value. Other authorities express the DFT and its inverse as:

$$X(k) = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x(n)\, e^{-2\pi jnk/N} \quad \text{DFT}$$

$$x(n) = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} X(k)\, e^{2\pi jnk/N} \quad \text{IDFT}$$

This last formulation is necessary if the power contents of the time-domain and frequency-domain signals are to be identical.

Throughout this application note, the definitions given by equations (6) and (8) are used for DFT and IDFT. However, the techniques described here are applicable to all three formulations of the DFT and its inverse.

III. ALGORITHMS FOR EFFICIENT EVALUATION OF DFT

From equation (6), it is apparent that the direct evaluation of the DFT is very much computation intensive. Assuming complex data, x(n)=xr(n) + j xi(n), we have

$$X(k) = X\,R(k) + jX\,I(k) =$$

$$= \sum_{n=0}^{N-1} [xr(n) + jxi(n)] \cos\left(\frac{2\pi nk}{N}\right) - j \sin\left(\frac{2\pi kn}{N}\right)$$

or

$$R(k) = \sum_{n=0}^{N-1} xr(n) \cos\left(\frac{2\pi nk}{N}\right) + xi(n) \sin\left(\frac{2\pi nk}{N}\right) \quad (9a)$$

and

$$I(k) = \sum_{n=0}^{N-1} xi(n) \cos\left(\frac{2\pi nk}{N}\right) - xi(n) \sin\left(\frac{2\pi nk}{N}\right) \quad (9b)$$

where k=0,1,2.....N-1, XR(k) and XI(k) are the real and imaginary parts of the spectrum respectively.

From equation (9) it can be deduced that the direct evaluation of DFT involves $4N^2$ multiplications and approximately $4N^2$ additions. In these estimates the computations involved in the evaluation of the trigonometric functions, (sin and cos) have been ignored as it is possible to precalculate the trigonometric values in a look-up table and use them appropriately. Historically, multiplications were very slow compared to other operations, therefore algorithms were developed to minimize the number of required multiplications at the expense of other operations. These algorithms made use of the cyclic nature of the exponential $\exp(\frac{-2\pi jnk}{N})$ to reduce the number of multiplications involved.

To demonstrate some of the resulting redundancies, consider the case where N is even. It is fairly straightforward to show that for this case

$$W_{\frac{N}{2}}^{k} = W_N^{2k} \quad (10)$$

and

$$W_N^k = -W_N^{k+\frac{N}{2}} \quad (11)$$

One algorithm which uses these types of redundancies is the Cooley-Tuckey radix-2 FFT (reference 1). This algorithm requires the number of data points to be equal to a power of two, ie $N=2^m$ where m is an integer. Using the identities given by (10) and (11) the algorithm expresses the N-point DFT in terms of two $\frac{N}{2}$-point DFT's. Then the $\frac{N}{2}$-point DFT's are expressed as two $\frac{N}{4}$-point transforms using identities similar to (10) and (11). This decomposition is carried out until all the DFT's involved are only two-point transforms. The net result of this decomposition is a considerable reduction in the number of multiplications. In fact it can be shown that for the Cooley-Tuckey radix-2 FFT algorithm the number of multiplications involved is approximately $2N \log_2(N)$ and the number of additions is $3N \log_2(N)$. Compared to the $4N^2$ operations involved in the direct DFT calculation, for a large N, the radix-2 FFT algorithm reduces the number of multiplications and additions considerably.

Another algorithm which also minimises the number of multiplications is Winogard's prime-length transform (reference 2). This algorithm is applicable to cases where the data size is a prime number. In practice this algorithm is used only for short-length transforms and mapping techniques are used to extend it to large data sizes (references 5 & 6).

The argument behind the efficiency of these algorithms is only valid if the multiplication time is longer than other operations such as indexing and memory accesses. This is indeed the case for most general-purpose processors.

Todays digital technology is capable of providing extremely powerful processing engines which mean that the minimization of the number of multiplications is not always the best approach. For high performance systems, other issues such as the memory bandwidth, architecture efficiency and parallelism potential have to be seriously considered. The advances in digital technology allow other algorithms particularly those which map the DFT onto special VLSI hardware structures to be exploited. The following sections deal with algorithms that map the DFT into correlation/convolutions, ideal for implementation using the IMSA100 transversal filter. This algorithms make use of the higher level functional nature of the device and its on-chip memory to minimise the required host's memory bandwidth. For this reason the combination of a medium-speed microprocessor and the IMSA100 device(s) results in a very high performance system capable of competing with bit-slice DSP processors.

IV. DFT ALGORITHMS SUITABLE FOR THE IMSA100 IMPLEMENTATION

There are basically two algorithms which map the DFT into a correlation (convolution) process. These are
(i) the Prime Number Transform (PNT)
(ii) the Chirp-Z-Transform (CZT)

The PNT was developed by Radar and is applicable when the number of data points is prime. The CZT on the other hand is applicable to any data size; it can, however, be simplified if the data size is an even number. The following two sections deal with each one of these algorithms and their implementations using the IMSA100 transversal filter. The final part of this application note describes mapping techniques which allow the DFT of a large number of data points to be evaluated via a number of short transforms. This mapping technique is of vital practical significance when implemanting PNT & CZT processors.

IV.1. Rader's Prime Number Transform

The PNT algorithm has its origin in number theory (reference 7) and consists of three seperate operations. The first is a permutation (re-ordering) of the input data. The second operation is correlation of the permuted input data with permuted discrete cosine and sine samples. The third operation is a repermutation, which yields the DFT components in the conventional order of linear frequency. This final stage may be ignored in applications which also involve an inverse DFT.

In this section the mathematical background for the PNT will be summarized. Where necessary examples are provided to assist in the understanding of the concepts.

If the standard DFT equation, i.e.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-2\pi jnk/N} = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \quad (12)$$

$$k = 0, 1, \ldots N-1$$

is to be converted to correlation between x(n) and the twiddle factors W_N's, the nk product needs to be converted to a sum n+k. For cases where N is prime number theory allows us to achieve this.

According to number theory (reference 7), for each prime number N, there exist integers r, known as primitive roots, whose successive integer powers modulo-N will generate a permuted version of the sequence 1, 2, 3,.....N-1.

What this means is that for a prime number N, it is possible to map the sequence {p}=0, 1, 2, ... N-2 via the equation

$$q=(r^q) \bmod N \text{ where } \{p\}= \{ 0,1,2,3,.....N\text{-}2 \} \quad (13)$$

to a sequence {q} where q is a one-to-one map of the original sequence {p} and consists of a permuted version of the sequence { 1, 2, 3,...N-1 }. For such a unique map to be possible r must be a primitive root of N. Let us consider N=7, for which one of the primitive roots of 7 is 3. From (13) the mapping equation is

$$q=(3^p) \bmod N \text{ where } p=0,1,2,....,5$$

For p=3 , q=(27)mod 7=6. Table 1 gives the corresponding values of p and q which confirm the one-to-one nature of the mapping.

It should be emphasized that for any prime N, the primitive root r, is not unique. For example Table 2 illustrates the mapping given by (13) for N=7. From

this table you can see that the mapping is unique and cyclic for r=3 and r=5 which are the primitive roots of 7. In most practical cases the smallest primitive root is often selected.

Table 1 : The mapping corresponding to equation (13) for r=3

p	0	1	2	3	4	5	6	7
q	1	3	2	6	4	5	1	3

Table 2 : Values for q in the mapping $q=(r^p) \bmod 7$

r\p	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	1	2	4	1	2	4	1	2	4	1
3*	1	3	2	6	4	5	1	3	2	6	4	5	1
4	1	4	2	1	4	2	1	4	2	1	4	2	1
5*	1	5	4	6	2	3	1	5	4	6	2	3	1
6	1	6	1	6	1	6	1	6	1	6	1	6	1
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	2	4	1	2	4	1	2	4	1	2	4	1
10	1	3	2	6	4	5	1	3	2	6	4	5	1
11	1	4	2	1	4	2	1	4	2	1	4	2	1
12	1	5	4	6	2	3	1	5	4	6	2	3	1
13	1	6	1	6	1	6	1	6	1	6	1	6	1

* Note that r=3 and r=5 give unique mapping and are therefore the primitive roots of 7.

If N is prime and r is primitive root of N then we would like to apply the mapping given by (13) to an N-point DFT. Referring to the DFT equation (12), it can be seen that the subscripts n and k vary from 0 to N-1 whilst in the mapping given by (13) the variable q assumes values from 1 to N-1 i.e it excludes zero. To overcome this problem we write the DFT equation (12) in the following form

$$X(0) = \sum_{n=0}^{N-1} x(n) \qquad (14a)$$

$$X(k) - x(0) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \quad k = 1, \ldots, N-1 \quad (14b)$$

i.e. we separate the expressions for the zero-frequency DFT component X(0) (the d.c. term) which is very simple. This expression consists of N additions only and if required can be calculated directly.

We are left with DFT components X(k) corresponding, to k=1, 2, . .N-1, which are given by (14b). Note that in this equation we have taken x(0) to the left-hand side so as the summation over n is from n=1 to n=N-1. We can now apply the mapping given by (13) to equation (14) via the following transformations,

$$n=(r^m) \bmod N \; m=0,1,\ldots\ldots,N-2 \quad (15)$$
$$k=(r^l) \bmod N \; l=0,1,\ldots\ldots,N-2 \quad (16)$$

which result in a permutation of the terms in the summation and a change in the order of the equations. Equation (14b) then becomes:

$$X[(r^l) \bmod N] - x(0) = \sum_{m=0}^{N-2} x[(r^m) \bmod N]\, W_N^{[(r^m) \bmod N]\,[(r^l) \bmod N]} \qquad (17)$$

where l=0,1,2,.....N-2

Remember that the twiddle factor, W_N, is cyclic in N, therefore we have

$$X[(r^l) \bmod N] - x(0) = \sum_{m=0}^{N-2} x[(r^m) \bmod N]\, W_N^{(m+l)} \qquad (18)$$

where l=0,1,2,.....N-2 This equation indicates that the sequence $X[(r^j) mod N]-x(0)$ can be calculated via a circular correlation of the permuted input sequence $x[(r^j) mod N]$ and the sequence $e^{-2\pi j r^j / N}$. To see this clearly let us consider in detail an example for a DFT of length 7.

The expression for a 7-point DFT is given by

$$X(k)=\sum_{n=0}^{6} x(n)\, W_N^{nk} = \sum_{n=0}^{6} x(n)\, e^{-2\pi j n k / 7} \qquad (19)$$

where n,k=0,1,2,.....N-1

This DFT equation can be expressed in matrix form as: (The subscript 7 has been dropped from W_7 for convenience)

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \end{bmatrix} = \begin{bmatrix} W^0 & W0 & W^0 & W0 & W^0 & W^0 & W^0 \\ W^0 & W1 & W^2 & W3 & W^4 & W^5 & W^6 \\ W^0 & W2 & W^4 & W6 & W^1 & W^3 & W^5 \\ W^0 & W3 & W^6 & W^2 & W^5 & W^1 & W^4 \\ W^0 & W4 & W^1 & W^5 & W^2 & W^6 & W^3 \\ W^0 & W5 & W^3 & W^1 & W^6 & W^4 & W^2 \\ W^0 & W6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \end{bmatrix} \qquad (20)$$

The superscript in each W^{nk} is evaluated mod 6. Noting that $W^0=1$ and separating the equation for X(0) we can rewrite equation (20) as:

$$\begin{bmatrix} X(1)-x(0) \\ X(2)-x(0) \\ X(3)-x(0) \\ X(4)-x(0) \\ X(5)-x(0) \\ X(6)-x(0) \end{bmatrix} = \begin{bmatrix} W^1 & W2 & W^3 & W^4 & W^5 & W^6 \\ W^2 & W4 & W^6 & W^1 & W^3 & W^5 \\ W^3 & W6 & W^2 & W^5 & W^1 & W^4 \\ W^4 & W1 & W^5 & W^2 & W^6 & W^3 \\ W^5 & W3 & W^1 & W^6 & W^4 & W^2 \\ W^6 & W5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \end{bmatrix} \quad (21)$$

and

$$X(0)=\sum_{n=0}^{6} x(n) \quad (22)$$

The expression for X(0) i.e equation (22) is a simple summation and is assumed to be evaluated separately.

Dealing with the computationally intensive part of the transform i.e. equation (21), we can apply the mapping given by (13) to this equation which would convert equation 21 into a cyclic correlation suitable for implementation using IMS A100 transversal filter. We choose r=3 which is the smallest primitive root of 7. The mapping would thus correspond to that given by Table 1. We first apply the permutation given by this mapping to the input sequence of x(n)'s. This would correspond to a column permutation of the twiddle matrix as shown in (23).

$$\begin{bmatrix} X(1)-x(0) \\ X(2)-x(0) \\ X(3)-x(0) \\ X(4)-x(0) \\ X(5)-x(0) \\ X(6)-x(0) \end{bmatrix} = \begin{bmatrix} W^1 & W3 & W^2 & W^6 & W^4 & W^5 \\ W^2 & W6 & W^4 & W^5 & W^1 & W^3 \\ W^3 & W2 & W^6 & W^4 & W^5 & W^1 \\ W^4 & W5 & W^1 & W^3 & W^2 & W^6 \\ W^5 & W1 & W^3 & W^2 & W^6 & W^4 \\ W^6 & W4 & W^5 & W^1 & W^3 & W^2 \end{bmatrix} \begin{bmatrix} x(1) \\ x(3) \\ x(2) \\ x(6) \\ x(4) \\ x(5) \end{bmatrix} \quad (23)$$

Note that the matrix equations (23) and (21) are essentially the same and their difference is only in the order of the terms. Next we apply the same mapping to the column matrix containing X(k)-x(0) terms in equation (23), this would of course correspond to a similar permutation of the rows of twiddle matrix in (23); and the result is given as equation (24).

$$\begin{bmatrix} X(1)-x(0) \\ X(3)-x(0) \\ X(2)-x(0) \\ X(6)-x(0) \\ X(4)-x(0) \\ X(5)-x(0) \end{bmatrix} = \begin{bmatrix} W^1 & W3 & W^2 & W^6 & W^4 & W^5 \\ W^3 & W2 & W^6 & W^4 & W^5 & W^1 \\ W^2 & W6 & W^4 & W^5 & W^1 & W^3 \\ W^6 & W4 & W^5 & W^1 & W^3 & W^2 \\ W^4 & W5 & W^1 & W^3 & W^2 & W^6 \\ W^5 & W1 & W^3 & W^2 & W^6 & W^4 \end{bmatrix} \begin{bmatrix} x(1) \\ x(3) \\ x(2) \\ x(6) \\ x(4) \\ x(5) \end{bmatrix} \quad (24)$$

Referring to equation (24) it can be seen that the twiddle-factor matrix has the property that each row can be obtained by a left-shift (rotate) of the previous row. This means that the sequence {X(1)-x(0), X(3)-x(0), X(2)-x(0), X(6)-x(0), X(4)-x(0), X(5)-x(0)} can be obtained by performing a circular convolution between the sequence {x(1),x(3),x(2)x(6),x(4),x(5)} and a permuted twiddle factor set given by {W^1, W^3, W^2, W^6, W^4, W^5}.

Figure 3 shows how this circular convolution can be implemented using a transversal filter structure. For the moment let us confine our attention to the canonical transversal filter structure and assume that the transversal filter is capable of complex processing i.e. both input data x(n) and the twiddle factors are complex. It will be shown later how a single IMSA100 device can be used to implement this complex processing. Two implementations are shown in Figure 3.

In the first implementation, Figure 3a, the permuted twiddle factors are used as the inputs to the transversal filter. These twiddle factors are first loaded into the filter with the input switch at position 1 and then circulated with the input switch at position 2. The output samples, as shown in Figure 3a, would correspond to the DFT of the input sequence x(n). You should be able to confirm this by referring to the matrix equation given by (24). For the arrangement in Figure 3a the allocation of the data sequence x(n) to the coefficient memory can be formulated as:

$$C(i) = x[(r^{N-2-i})\text{mod}N] \quad i = 0, 1, ..., N-2 \quad (25)$$

where N is 7 in this case. Similarly the twiddle factor sequencing, at the input of Figure 3a, can be mathematically expressed as:

$$\text{Input}(i) = W^{[(r)^i \text{mod}N]} \quad i = 0, 1, ..., N-2 \quad (26)$$

The output sequence for Figure 3a is given by:

$$\text{Output}(i) = X[(r^i)\text{mod}N] \quad i = 0, 1, ..., N-2 \quad (27)$$

Figure 3b shows the second possible implementation in which the coefficient memory contains the permuted twiddle factors and the permuted data sequence is loaded at the input of the filter and is circulated to generate the DFT of the input samples. For this implementation the generalized equations for the input and output sequences and the allocation of coefficient memory are:

$$C(i) = W^{[(r^i)\text{mod}N]} \quad i = 0, 1, ..., N-2 \quad (28)$$

$$\text{Input}(i) = x[(r^{N-2-i}\text{mod}N] \quad i = 0, 1, ..., N-2 \quad (29)$$

$$\text{Output}(i) = X[(r^i)\text{mod}N] \quad i = 0, 1, ..., N-2 \quad (30)$$

Equation (25) to (27) and (28) to (30) define the required permutation and sequencing for a generalised prime number transform based on the canonical transversal filter structure.

It was argued earlier that the IMSA100 implementation of the transversal filter structure (Figure 3b) is identical in behaviour to that of the canonical form (Figure 3a). The only difference is that in the canonical form the first coefficient, C(0), is associated with the left most memory location (see Figure 3a) while in the IMSA100 implementation the right most coefficient register is allocated to C(0). We will now show how our 7-Point DFT can be implemented in complex form using the IMSA100 transversal filters. The input data samples x(n), the twiddle factors W^n, and the DFT output samples X(n) can be expressed in terms of their real and imaginary parts as:

$$x(n) = xr(n) + j\,xi(n) \quad (31)$$

$$W_N^n = e^{-2\pi/N} = \cos\left(\frac{2\pi n}{N}\right) + j\sin\left(\frac{-2\pi n}{N}\right) = WR(n) + j\,WI(n) \quad (32)$$

$$X(n) = XR(n) + j\,XI(n) \quad (33)$$

As mentioned earlier the IMSA100 device contains two sets of coefficient memories; at any instant one set of the coefficients is used in the computation whilst the other set can be accessed via a standard memory interface. One very important feature of the device is that the two memory banks can be exchanged automatically at the beginning of every computation cycle. i.e. alternate set of coefficients are applied to the filter successively. This feature allows complex convolution and correlation to be performed in a single device. (This is unlike most conventional realizations of complex convolution/correlation where, as shown in Figure 4, four transversal filters are often used to implement these complex functions). This is achieved by appropriately loading the two coefficient memories with combinations of real and imaginary samples of the reference signal and using the continuous memory-swap mode to implement complex processing. The real and imaginary parts of the signal to be correlated (or convolved) with the reference signal are then applied alternatively to the input of the IMSA100 device. An application note entitled 'Complex Processing Using the IMSA100 Transversal Filters' covers this topic in detail and is available from INMOS. The remainder of this section gives an overview of the topic in relation to complex DFTs.

Figure 3 : Prime Number Transform Implementation Based on the Transversal Filter Structures

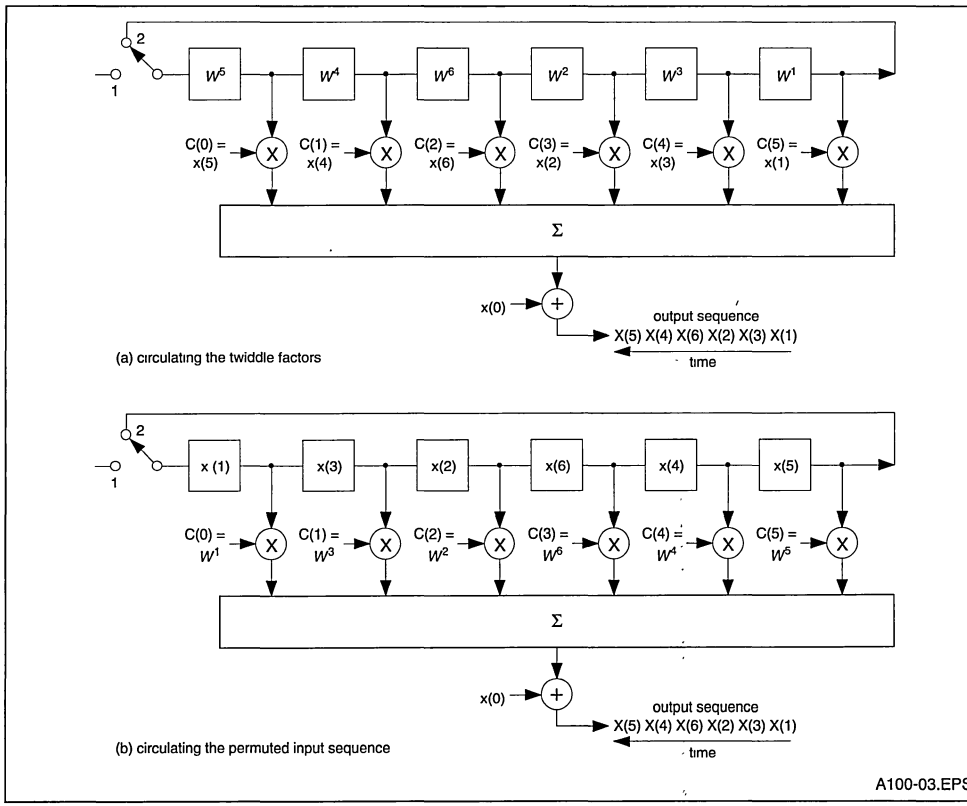


Figure 4 : Conventional Complex Correlator/convolver Involiving Four Transversal Filters

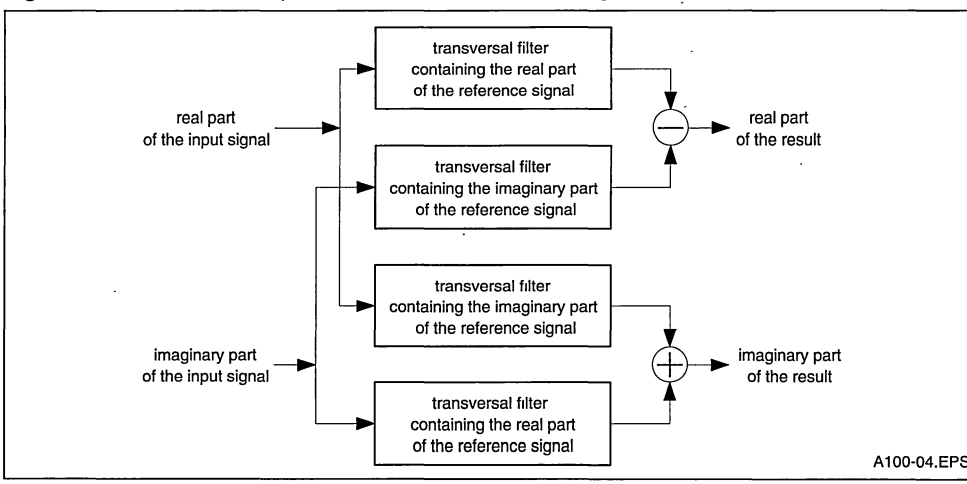


Figure 5 shows the IMSA100 implementation of the 7-point DFT example, corresponding to Figure 3a, where the twiddle factors are circulated. The notation used for real and imaginary parts of the signals is that given by equations (31) to (33). The twiddle factors are applied to the input in a sequence identical to that used in Figure 3a, with the real part of each sample followed by its imaginary part. The output sequence is also shown. It is assumed that the delay-and-add chain in the IMSA100 is cleared first by writing several zero's to the input. (Note that this is only needed once and any further transforms do not need this flushing). The memory banks are set in their continuous-swap mode. In the first computation cycle, with WR(1) on the input, the memory bank 'A' is used in the computation; in the second cycle, when WI(1) is applied to the input, the memory bank 'B' is used in the computation; in the third cycle WR(3) is the input sample and the memory bank A is used in the computation and so on. Note also that for each output sample an external addition (with either xr(0) or xi(0) depending on whether the output corresponds to real or imaginary part of the result) has to be carried out as dictated by equation (24). This is a negligible overhead compared to the computation performed by the transversal filter and can easily be carried out by the host processor.

In Figure 5 the first eleven output samples (denoted with *) are partial results and as such are not fully valid. This is due to the inherent delay associated with any transversal filter implementation. In continuous processing, however, it is possible to avoid these undefined output samples and to achieve a duty cycle of 100% by updating two coefficient memory locations with new data samples. For example in Figure 5, assumming that we have applied the first cycle of the twiddle factor values i.e. WR(1), WI(1), WR(3), WI(3) . . . . . WR(5), WI(5) to the input, it is possible to update the coefficient locations corresponding to xi(1) and xr(1) in memory bank A with the imaginary and real parts of the first sample of the new input data block. This can be done during the latest computation cycle (with WI(5) as the input twiddle factor) when the memory bank A is free. In the next cycle when WR(1) is applied to the input, xr(1) and -xi(1) in the memory bank B can be updated with new data values while this memory bank is free. In the following computation cycle when WI(1) is the input twiddle factor, the coefficient memory locations corresponding to xi(3) and xr(3) in the memory bank A are updated and so on. This technique removes the undefined output samples and achieves a duty cycle of 100%.

Figure 6 depicts the IMSA100 implementation of the 7-point DFT, corresponding to Figure 3b, where the input data samples are recirculated and the twiddle factors are stored in the coefficient memories. The input data samples are applied to the input in a sequence identical to that used in Figure 3b, with real part of each sample followed by its imaginary part. The output sequence is also shown. Other characteristics of this implementation are identical to the previous case with the exception that in this implementation it is impossible to avoid initial undefined output samples even when several continuous transforms are to be performed.

The IMSA100 devices can be cascaded without any external components by simply connecting the output of the first device to the cascade-input of the second device. This simple cascading allows transversal filters/correlators with many stages to be easily implemented.

Using prime-number algorithm there are basically two ways to implement A100 based DFT processors capable of handling long data blocks. The obvious approach is to cascade several devices resulting in a sufficiently large correlator/convolver capable of dealing with the whole data block size. This approach is only acceptable for moderate block sizes and becomes impractical if the data size is very large. The second approach is based on mapping techniques which convert a large DFT into several independent short transforms. These short transforms can then be evaluated either concurrently or sequentially, depending on the required performance. This means that the decomposition techniques described here are particularly useful as they provide the basis for trade-offs between cost and speed. This subject will be discussed in detail in section 5 of this application note.

Figure 5 : IMSA100 implementation of a 7 point complex DFT, corresponding to the canonical transversal filter realization of Figure 3a

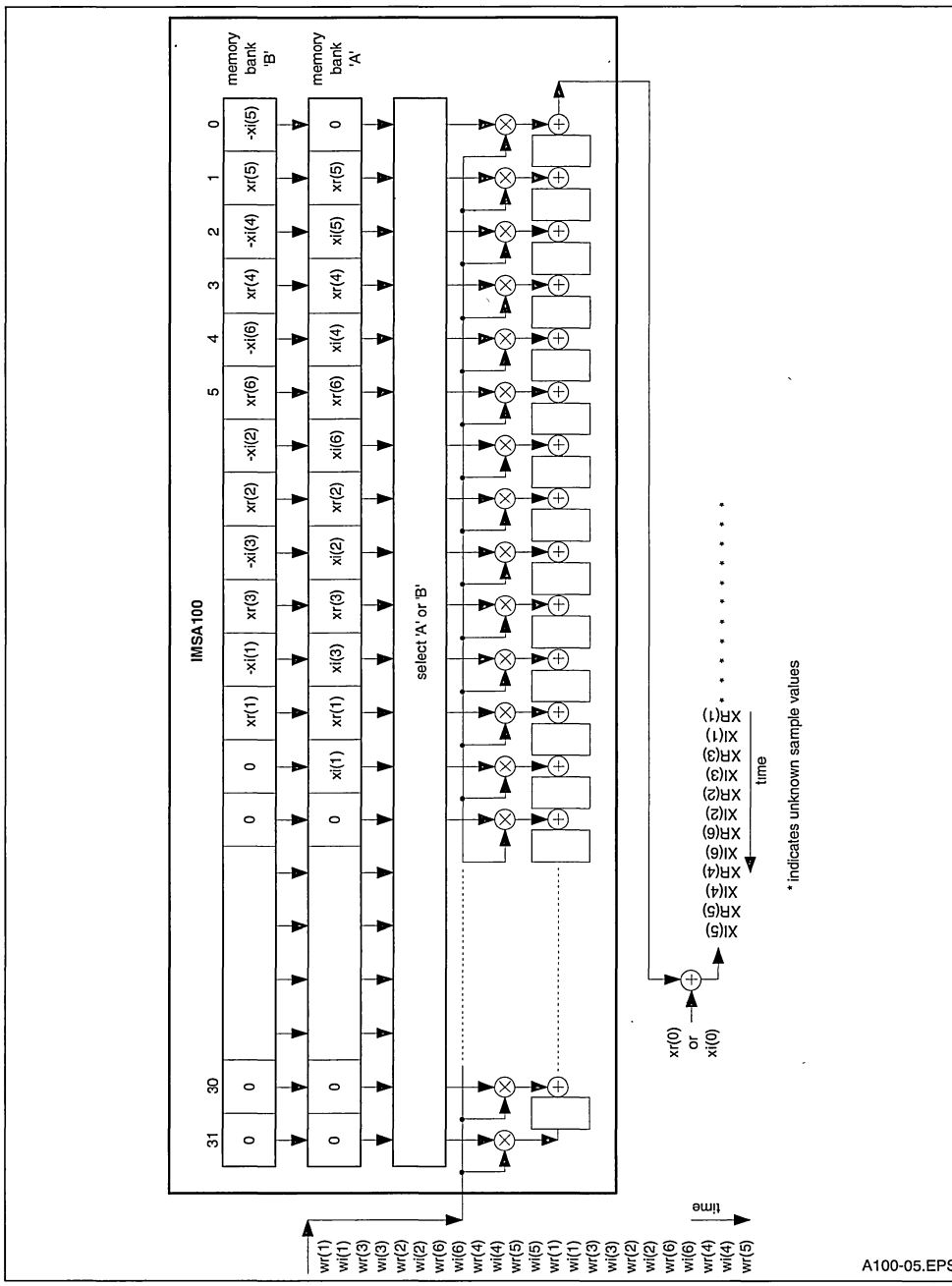
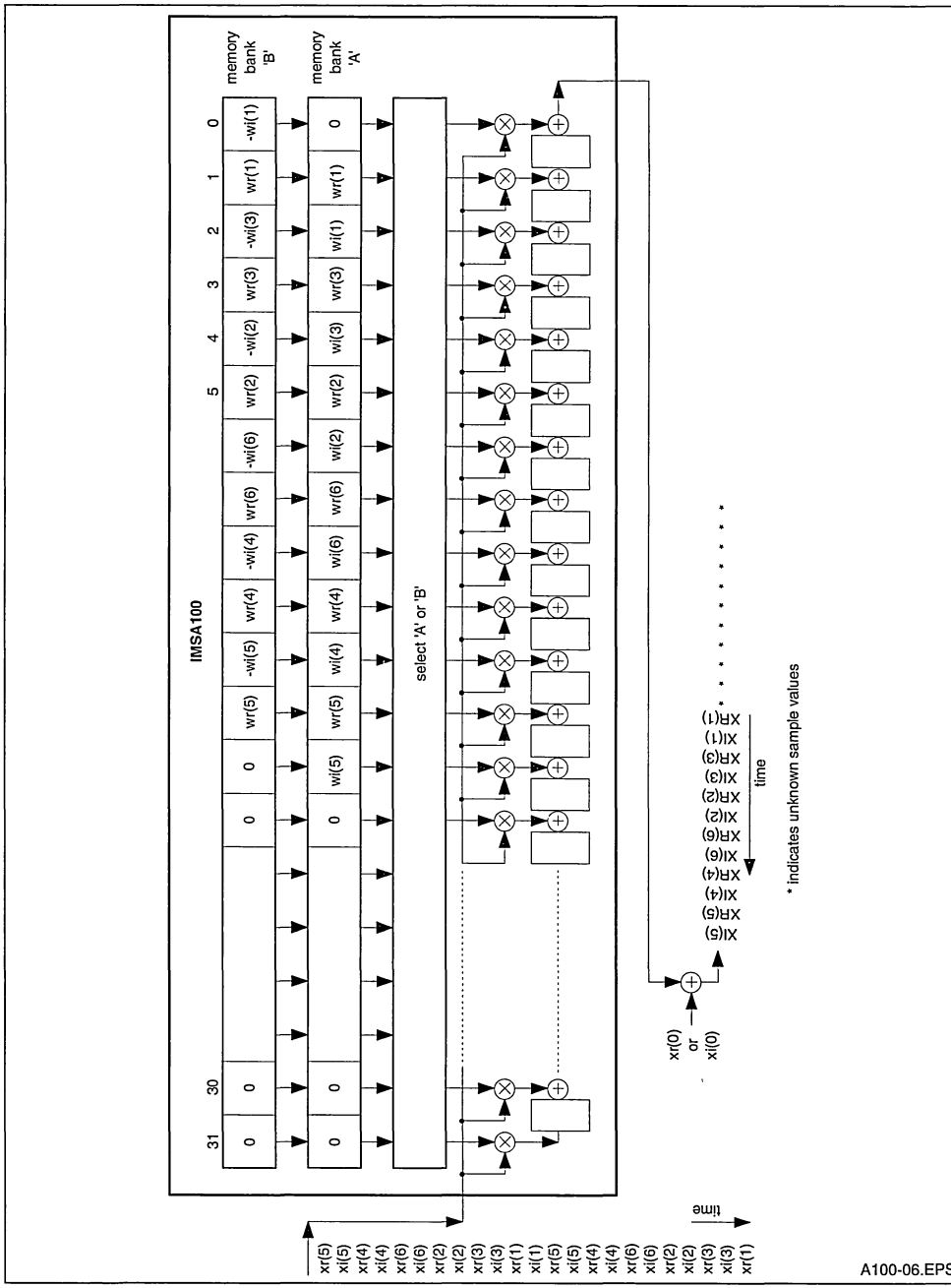


Figure 6 : IMSA100 implementation of a 7 point complex DFT, corresponding to the canonical transversal filter realization of Figure 3b



As mentioned earlier the IMSA100 transversal filter has an on-chip industry standard memory interface which allows the part to be fully memory-mapable. Figure 7 shows a schematic diagram of a simple system making use of this memory interface. When implementing the prime transform algorithm on this system, the IMSA100 (or arrays of them) will perform the bulk of the computation and the host processor will be responsible for data permutation (using look-up tables), evaluating the X(0) term (equation 22), and performing the auxiliary addition involving either xr(0) or xi(0) (see Figures 5 & 6).

Another possible system configuration is shown in Figure 8. This is particularly suitable for the arrangement of Figure 5. The real and imaginary parts of the twiddle factors are pre-loaded in the memory MEM1 and are supplied to the A100 via the dedicted input port. The sequencer shown in Figure 8 could be a simple counter. The processor accesses the coefficient memories and the output result via the IMSA100's memory interface. Other system configurations are possible. For example Figure 9 shows the schematic of a high performance signal processing system using a dedicated controller.

Figure 7 : Schematic Diagram of a Simple IMSA100 Based System

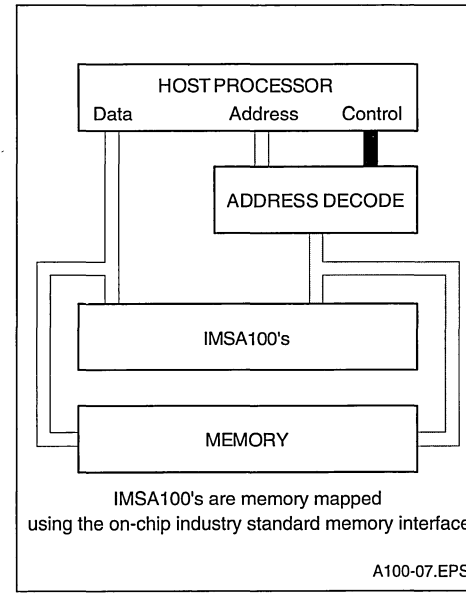


Figure 8 : An Implementation Particularly Suitable for the Arrangement in Figure 5

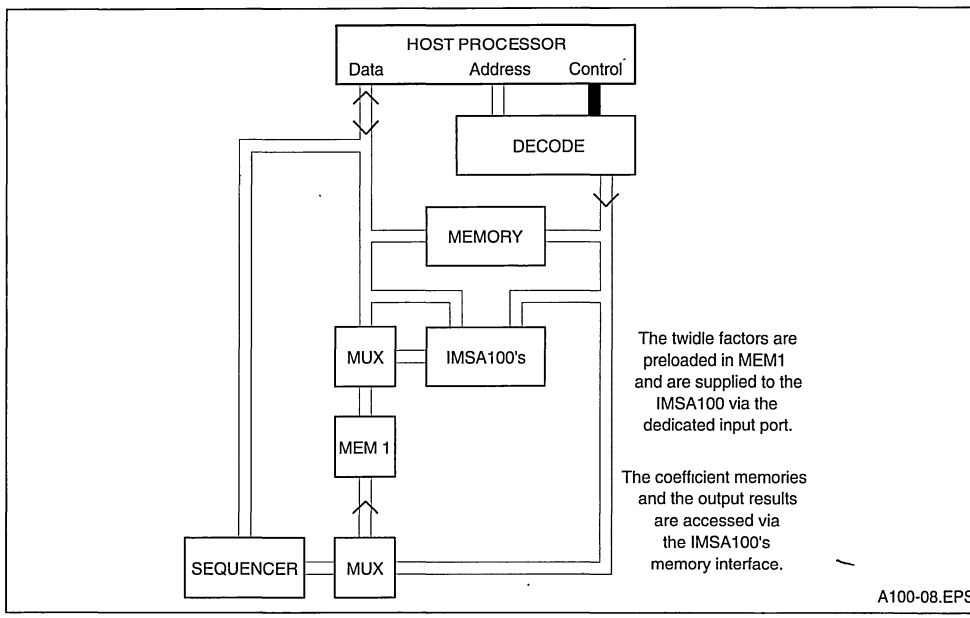
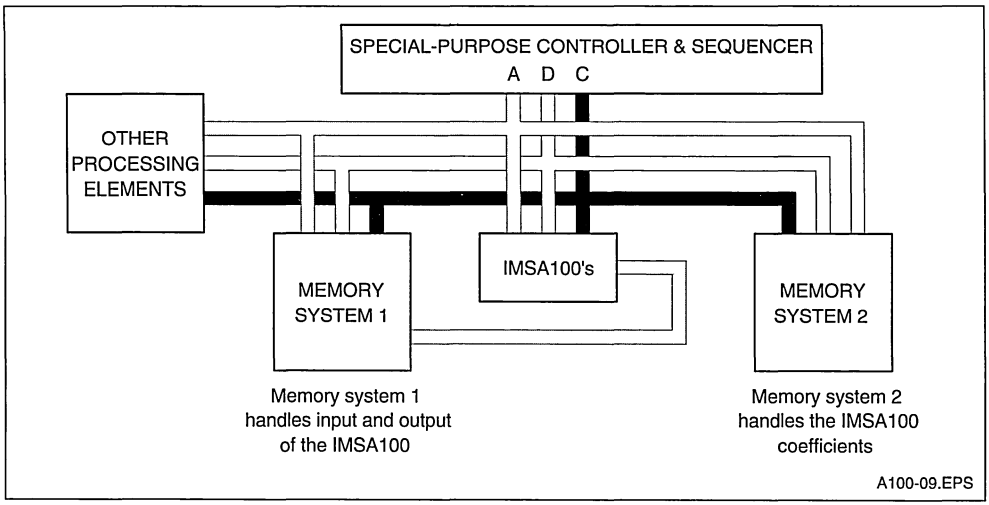


Figure 9 : Schematic Block Diagram of a High Performance Involving a Special Purpose Controller



IV.2. The chirp-z transform

Another algorithm which converts the DFT equation into a convolution (or correlation) is the chirp-z transform. The reason for the name chirp is that the transform uses a sampled linear frequency-modulated carrier which in signal processing is often termed a 'chirp signal'. In the previous section we saw that the prime number transform consisted of three operations, namely

(i) Data input permutation.

(ii) Convolution (or correlation) of the permuted data with a permuted sequence of twiddle factors.

(iii) A final permutation to obtain the correct output sequence.

Figure 10 summarizes the principles of the prime number transform algorithm. The auxillary computation for zeroth input sample and evaluation of X(0) are also shown in this schematic diagram. The structure of the chirp-z DFT algorithm is similar to that of the prime number transform technique and consists of the following sequence of three operations:

(i) Premultiplication of the input sequence x(n) by the chirp $e^{(-\pi jn^2/N)}$.

(ii) Convolution (or correlation) of the resulting sequence with a second chirp signal.

(iii) Post-multiplication of the resulting sequence by the chirp signal $e^{(-\pi jk^2/N)}$.

These operations are summarized in Figure 11. Comparing Figure 10 and Figure 11, it can be seen that the major difference between the two algorithms is that in the chirp-z transform the permutation operations are replaced by multiplications.

Figure 10 : The Principle of the Prime Number Transform

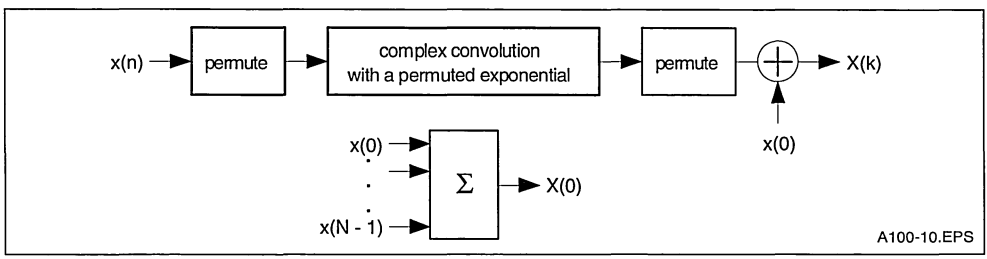
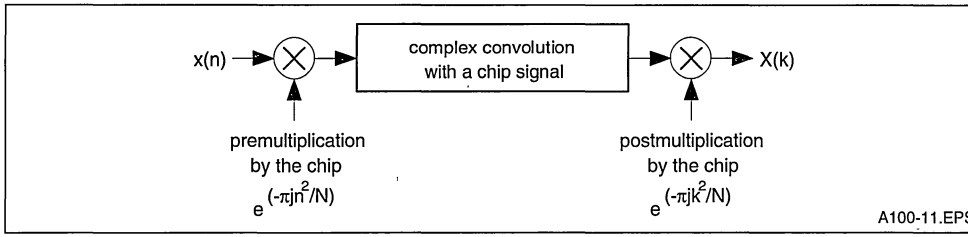


Figure 11 : The Basic Principle of the CZT



In the CZT the convolution (correlation) operations can be implemented using the IMSA100 transversal filter, but the pre- and post-multiplications have to be done externally. In many applications data permutation may be preferred to multiplication in which case the prime numbers approach may be considered more advantageous. However there are applications (e.g beamforming) where the CZT, in particular a simplified version of it referred to as sliding CZT, is preferred.

To understand the CZT algorithm we start from the DFT equation

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-2j\pi nk/N} = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \quad (34)$$

$$k = 0, 1, \ldots N-1$$

and replace the term -2nk in the complex exponential with the seemingly more complicated expression

$$-2nk = (k-n)^2 - k^2 - n^2 \quad (35)$$

hence

$$X(k) = e^{-j\pi k^2/N} \sum_{n=0}^{N-1} [x(n)\, e^{-j\pi n^2/N}][e^{j\pi(k-n)^2/N}] \quad (36)$$

$$= e^{-j\pi k^2/N} \sum_{n=0}^{N-1} y(n)\, [e^{j\pi(k-n)^2/N}]$$

where k=0,1,.....,N-1.

In equation 36, the term X(k) consists of three operations:

(i) Multiplications of the samples X(n) with a complex linear frequency-modulated signal $e^{-j\pi n^2/N}$ to form a new set of samples y(n); This operation is often referred to as premultiplication by a chirp,

(ii) the convolution of y(n) with a second-linear frequency-modulated signal (the term $e^{j\pi(k-n)^2/N}$ and

(iii) post multiplication by $e^{-j\pi k^2/N}$.

Note that if only the power spectrum of the signal is required the final operation can be omitted, since $e^{-j\pi k^2/N}$ represent only a phase-shift and $|e^{-j\pi k^2/N}|=1$. Also in operation (ii) the term $(k-n)^2$ in the complex exponential is equal to $(n-k)^2$ so that a convolution operation, in this case, is identical to that of a correlation. Figures 12 and 13 show examples for a 6-point CZT implemented using the canonical transversal filter structure. In these diagrams it is assumed that the transversal filter is capable of complex processing. As described in the previous section the complex conolution/correlation can easily be implemented using the IMSA100 transversal filter chip.

In Figure 12, samples corresponding to the product of the input data and the premultiplying chirp are stored as the filter coefficients and a second sampled chirp is fed to the input of the convolver. Note that the convolver has N-complex points. Figure 13 shows an alternative implementation where the product of the input data samples and the premultiplying chirp samples are the inputs to the convolver and a chirp signal is used as the reference signal in the coefficient store. Note that in this arrangement the convolver has to have 2N-1 complex points. However when the number of points in the transform are even (N=even), it can easily be shown that the sampled chirp signal $f(n) = e^{j\pi n^2/N}$ has the following properties:

(i) it is periodic with a periodicity equal to N i.e.

$$f(n) = f(n+N) \quad (37)$$

(ii) It is symmetrical about $\frac{n=N}{2}$ i.e.

$$f(n) = f(N-n) \quad (38)$$

These properties convert the convolution in Figure 13 into a circular one which can be implemented with an N-point complex transversal filter.

In many applications the PNT may be preferred to the CZT because of the requirement of pre- and post-multiplications in the latter. (Remember that in the PNT, permutation operations are involved rather than multiplications).

As there are considerable amount of literature available on CZT, it will not be considered here in any more detail.

Figure 12 : Schematic Diagram for 6-point CZT using canonical Transversal Filter Structure

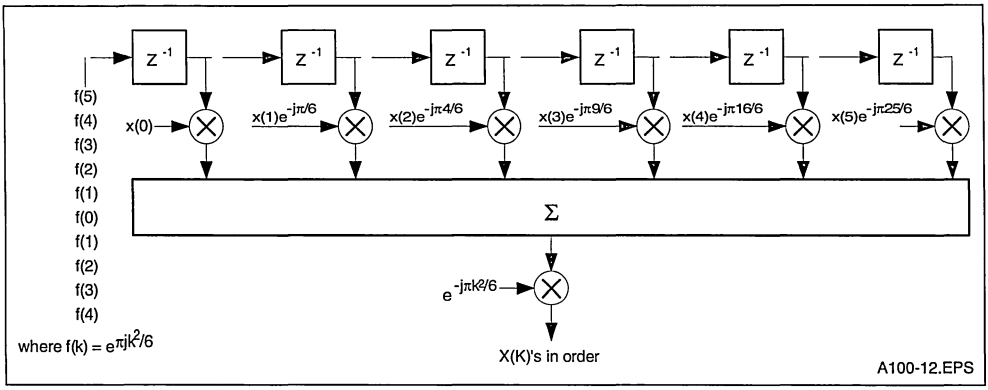
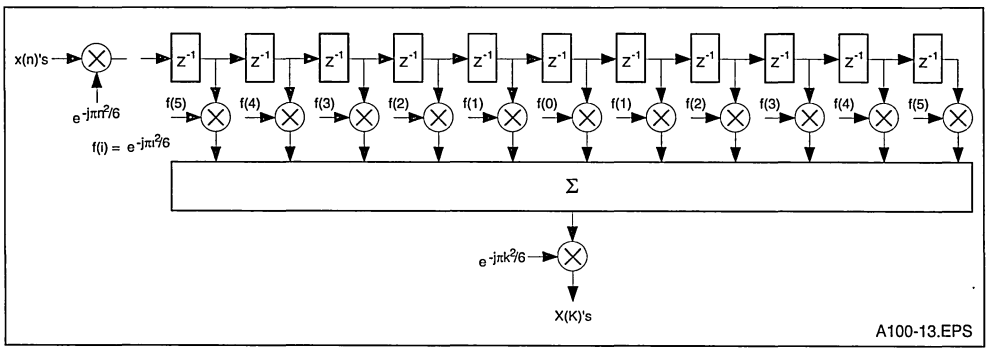


Figure 13 : Alternative Implementation of the 6-point CZT



V. MULTIDIMENSIONAL INDEX MAPPING FOR DFT DECOMPOSITION

In the previous sections, algorithms which convert the DFT into convolution/correlation operations were presented. These algorithms, particularly the PNT, are suitable for implementation using the IMSA100 transversal filter.

In order to compute the DFT of long data sequences, one approach is to cascade several the IMSA100 devices so that sufficient convolution/correlation points are made available. This approach is only acceptable for moderate data sizes, and does not provide the optimum performance for a given number of devices.

In this section, index mapping techniques are described which allow long DFT's to be decomposed into several shorter transforms. These shorter transforms can then be efficiently implemented by using IMSA100 transversal filters. The decomposition techniques described here can be viewed as generalised algorithms, with radix-2 FFT being a special case of these more general partitioning techniques. These mapping techniques provide the basis for designing highly concurrent systems and optimization in terms of performance and cost.

V.1. Basic concepts of index mapping

The essence of these mapping techniques is that by a simple change of variable, the original complex problem is converted into several easy ones. Before applying these techniques to the DFT, let us consider a few examples which should help to familiarize the reader with the terminologies used in general index mapping.

Consider a one-dimensional array of data x(n), n=0 to N-1, where N is the total number of elements in the array. For N=6, the array elements will be given by {x(0) x(1) x(2) x(3) x(4) x(5)}

Let us rearrange this one-dimensional array into a two-dimensional array as shown below:

$$[\ x(0)\ \ x(1)\ \ x(2)\ \ x(3)\ \ x(4)\ \ x(5)\ ] \overset{\text{map}}{\Leftrightarrow} \begin{bmatrix} x(0) & x(1) & x(2) \\ x(3) & x(4) & x(5) \end{bmatrix} = \begin{bmatrix} y(0,0) & y(0,1) & y(0,2) \\ y(1,0) & y(1,1) & y(1,2) \end{bmatrix} \quad (39)$$

We have 'mapped' the original one-dimensional array, x(n) , into a two-dimentional array $y(n_1, n_2)$. It can be seen that in this example the mapping is given by the following linear equation.

$$n = 3n_1 + n_2\ ,\ x(n) = y\ (n_1 + n_2) \quad (40)$$

where n_1=0, 1 and n_2=0, 1, 2.

The mapping is said to be one-to-one (unique) as all the elements in the original array, x(n) , appear in the two-dimensional array $y(n_1, n_2)$.

As a second example consider the following mapping:

$$[\ x(0)\ \ x(1)\ \ x(2)\ \ x(3)\ \ x(4)\ \ x(5)\ ] \overset{\text{map}}{\Leftrightarrow} \begin{bmatrix} x(0) & x(2) & x(4) \\ x(3) & x(5) & x(1) \end{bmatrix} = \begin{bmatrix} y(0,0) & y(0,1) & y(0,2) \\ y(1,0) & y(1,1) & y(1,2) \end{bmatrix} \quad (41)$$

This mapping has been obtained from

$$n = (3n_1 + 2n_2)\ \text{mod}6 \quad (42)$$

Note that in equation (42) the index n is evaluated modulo N (in this case N=6) and it is therefore cyclic in N.

As a final example let us apply the following mapping

$$n = (2n_1 + 2n_2)\ \text{mod}6 \quad (43)$$

to our 6-element array, which gives;

$$[\ x(0)\ \ x(1)\ \ x(2)\ \ x(3)\ \ x(4)\ \ x(5)\ ] \overset{\text{map}}{\Leftrightarrow} \begin{bmatrix} x(0) & x(2) & x(4) \\ x(2) & x(4) & x(0) \end{bmatrix} = \begin{bmatrix} y(0,0) & y(0,1) & y(0,2) \\ y(1,0) & y(1,1) & y(1,2) \end{bmatrix} \quad (44)$$

Obviously this map is not unique as x(1), x(3) and x(5) are not represented in the matrix $y(n_1, n_2)$.

V.2. Generalisation and conditions for uniqueness

Let us now generalize the ideas developed in the previous examples. We are interested in mapping a one-dimensional array of length $N=N_1 \times N_2$ into a two-demensional array that is N_1, by N_2 in size. In other words, the one-dimensional array

$$x(n)\ \text{for}\ n=0,1,\ldots N-1$$

is to be mapped into a two-dimensional array

$$y(n_1,n_2)\ \text{for}\ n_1=0,1,\ldots N_1-1,\ \text{and}\ n_2=0,1,\ldots N_2-1.$$

The major requirement is that the mapping must be unique. This mapping can be represented by

$$[\ x(0)\ \ x(1)\ \ x(2)\ \ \ldots\ \ x(N-1)\ ] \overset{\text{map}}{\Leftrightarrow} \begin{bmatrix} y(0,0) & y(0,1) & \ldots & y(0,N_2-1) \\ y(1,0) & y(1,1) & \ldots & y(1,N_2-1) \\ . & . & & . \\ . & . & \ldots & . \\ . & . & & . \\ y(N_1-1,0) & y(N_1,1) & \ldots & y(N_1-1,N_2-1) \end{bmatrix} \quad (45)$$

where

$$x(n) = y\ (n_1 + n_2) \quad (46)$$

This map, in general, can assume many different forms, but the one particularly useful to the DFT is the linear form,

$$n = (M_1 n_1 + M_2 n_2) \bmod N \quad (47)$$

Note that n is evaluated modulo N, making the map cyclic in n. In order for this map to be unique and one-to-one, the mapping constants M_1 and M_2 must satisfy certain conditions. In the literature (references 5 & 6) these conditions have been derived from number theory for two cases which are described in the following two subsections.

V.2.a. RELATIVELY PRIME CASE

In this case N_1 and N_2 are relatively prime and have no common factor. In the literature this case is often denoted by:

$$(N_1, N_2) = 1 \quad (48)$$

which means that the greatest common divisor of N_1 and N_2 is unity. For example (5,7)=1, (8,9)=1 and (6,25)=1. For this case the conditions on M_1 and M_2 which make the mapping, given by (47), unique and one-to-one are: (references 5 & 6)

$[(M_1 = \alpha N_2)$ and/or $(M_2 = \beta N_1)]$ and $(M_1, N_1) = (M_2, N_2) = 1 \quad (49)$

where a and b are integers.In other words, to ensure a unique maping for this case:

(M_1 must be a multiple of N_2)
or (M_2 must be a multiple of N_1)
or (M_1 and M_2 must be multiples of of N_2 and N_1 respectively)
and M_1 and N_1 must be relatively prime.
and M_2 and N_2 must be relatively prime.

As an example consider N_1=5, N_2=7, N=35. From (49) we have to choose M_1 a multiple of N_2 or M_2 a multiple of N_1 or both. Let us make M_1 the simplest multiple of N_2 i.e. M_1=a N_2=N_2=7, this also satisfies (M_1,N_1)=(7,5)=1. Then noting that we must have (M_2,N_2)=$(M_2,7)$=1, possible values for M_2 are: 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 15. While M_2=7, 14, 21 . . . are not allowed as they are not relatively prime with N_2. (Note also that for M_1=5, 10, 15, we also have M_2=a N_1, which is allowed.)

If M_1 is chosen to be M_1=2× N_2=14, then again the same values of M_2 as above are valid.

This example shows that a large class of unique mappings exist for this case.

V.2.b. COMMON FACTOR CASE

In this case N_1 and N_2 have a common factor r. i.e. their greatest common divisor is r and we have:

$$(N_1, N_2) = r \quad (50)$$

For example (10,5)=5, (9,12)=3, and (7,21)=7. For this case the conditions on M_1 and M_2, making the mapping given by (47) unique, have been shown to be: (references 5 & 6)

$(M_1 = \alpha N_2)$ and $(M_2 \neq \beta N_1)$ and $(\alpha, N_1) = (M_2, N_2) = 1 \quad (51a)$

or

$(M_1 \neq \alpha N_2)$ and $(M_2 = \beta N_1)$ and $(\beta, N_2) = (M_1, N_1) = 1 \quad (51b)$

where a and b are integers.

As an example consider N_1=9,N_2=15, N=135. From (51a) we can choose, M_1=a N_2=N_2=15. The condition (a ,N sub 1)=(1,9)=1 is already satisfied. From (51a), values of M_2 which are allowed are those which satisfy ($M_2 \neq b\times 9$) and (M_2,15)=1. Therefore following values of M_2 are allowed

M2=1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, 22, ...

M_2 = 3, 5, 6, 9, 10, . . . are not allowed as they do not satisfy (M_2,15)=1.

Alternatively we could have chosen M_1=a N_2=2× 15=30, then possible values of M_2 would again be

M_2=1, 2, 4,7, 8, 11, 13, 14, ...

However we could not have chosen M_1=a N_2=3× N_2 since this violates the requirement (a, N_1)=1.

V.3. Application of index mapping to DFT decomposition

Having covered the basic principle of index mapping and the required conditions for uniqueness, let us apply the mapping given by (47) to the DFT and investigate its consequences.

Remember that the DFT equation is given by

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-2\pi j nk/N} = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \quad (52)$$

$$k = 0, 1, \ldots, N-1$$

and that the exponent of W is evaluated modulo N since

$$W_N^{nk} = W_N^{nk+N} \quad (53)$$

Let us apply the following mappings to the indices n and k; i.e. to both the input data array x(n), and the result array X(k);

$$n = (M_1 n_1 + M_2 n_2) \bmod N \quad (54)$$
$$k = (L_1 k_1 + L_2 k_2) \bmod N \quad (55)$$

Where n_1, and k_1, are indexed to (N_1-1) and n_2 and k_2 to (N_2-1). As shown below these mappings convert the one dimensional arrays x(n) and X(k) into two-dimensional matrices $y(n_1, n_2)$ and $Y(n_1, n_2)$ respectively.

$$[x(0)x(1)...x(n)...x(N-1)] \quad \text{map} \Leftrightarrow \quad \begin{bmatrix} y(0,0) & y(0,1) & \cdots & \cdots & y(0,N_2-1) \\ y(1,0) & y(1,1) & \cdots & \cdots & y(1,N_2-1) \\ \vdots & \vdots & \ddots\vdots\ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots\ y(n_1,n_2) & \ddots & \vdots \\ \vdots & \vdots & \ddots\vdots\ddots & \ddots & \vdots \\ y(N_1-1,0) & y(N_1,1) & \cdots & \cdots & y(N_1-1,N_2-1) \end{bmatrix}$$

where $y(n_1,n_2)=x(n)$.

And

$$[X(0)X(1)...X(n)...X(N-1)] \quad \text{map} \Leftrightarrow \quad \begin{bmatrix} Y(0,0) & Y(0,1) & \cdots & \cdots & Y(0,N_2-1) \\ Y(1,0) & Y(1,1) & \cdots & \cdots & Y(1,N_2-1) \\ \vdots & \vdots & \ddots\vdots\ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots\ Y(k_1,k_2) & \ddots & \vdots \\ \vdots & \vdots & \ddots\vdots\ddots & \ddots & \vdots \\ Y(N_1-1,0) & Y(N_1,1) & \cdots & \cdots & Y(N_1-1,N_2-1) \end{bmatrix}$$

where $Y(k_1,k_2)=X(k)$.

Applying these mappings to the DFT equation gives:

$$X(L_1 k_1 + L_2 k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(M_1 n_1 + M_2 n_2)\, W_N^{nk} \quad (56)$$

or

$$Y(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} y(n_1,n_2)\, W_N^{nk} \quad (57)$$

with

$$W_N^{nk} = W_N^{M_2L_2n_2k_2}\, W_N^{M_1L_2n_1k_2}\, W_N^{M_1L_1n_1k_1}\, W_N^{M_2L_1n_2k_1} \quad (58)$$

Let us now partially define the maps in (54) and (55) by setting;

$$M_2 = \beta N_1 \text{ and } L_1 = \gamma N_2 \quad (59)$$

Where β and γ are integers.

These assignments make the last term in (58) i.e $W_N^{M_2L_1n_2k_1}$ equals to unity.Let us now separately consider each one of the two cases studied earlier and investigate the effect on the three remaining terms in (58).

V.3.a. CASE 1 — PRIME FACTOR DECOMPOSITION

For this case N_1 and N_2 are assummed to be relatively prime. From (49) it can be seen that it is possible to also set:

$$M_1 = \alpha N_2 \text{ and } L_2 = \delta N_1 \quad (60)$$

This makes the second term in (58) i.e. $W_N^{M_1L_2n_1k_2}$ also equal to unity. The remaining two terms in (58) can be written as:

$$W_N^{M_2L_2N_2k_2} = W_N^{\beta N_1\delta N_1n_2k_2} = W_{N_2}^{\beta\delta N_1n_2k_2} \quad (61)$$

$$W_N^{M_1L_1N_1k_1} = W_N^{\alpha N_2\gamma N_2n_1k_1} = W_{N_1}^{\alpha\gamma N_2n_1k_1} \quad (62)$$

The DFT equation will therefore become:

$$Y(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1,n_2)\, W_{N_2}^{\beta\delta N_1n_2k_2} \right] W_{N_1}^{\alpha\gamma N_2n_1k_1} \quad (63)$$

where $k_1=0,1,2,...,N_1-1$ and $k_2=0,1,2,...,N_2-1$

The advantage of this equation is that it uncouples the DFT calculations in the sense that the N-point DFT can be mapped into two completely separate sets of short DFT's. In evaluating the $Y(k_1,k_2)$'s, in equation (63), the inner summations over n_2 are operations involving separate rows of the matrix $y(n_1, n_2)$. The outer summations over n_1, on the other hand, are column operations and can be carried out after the row operations are completed. By suitable choice of α, β, γ, and δ, each one of these summations can be expressed as a DFT of the corresponding row or column. Goods (reference 8) suggested:

$$\alpha = \beta = 1$$
$$\gamma = (N_2^{-1}) \text{mod} N_1 \quad (64)$$
$$\delta = (N_1^{-1}) \text{mod} N_2$$

[Note that in modulo arithmetic the reciprocal of a number (g) mod N is denoted by (g^{-1}) mod N and is defined as : $[(g) \text{ mod } N][(g^{-1} \text{ mod } N]=1$.

For example (3^{-1}) mod 7=(5) mod 7 since 5× 3=15 which is 1 modulo 7.]

Applying (64) to (63) gives:

$$Y(k_1,k_2) = \sum_{n_1=0}^{N_1-1}\left[\sum_{n_2=0}^{N_2-1} y(n_1,n_2)\, W_{N_2}^{n_2k_2}\right] W_{N_1}^{n_1k_1}$$

$$= \sum_{n_1=0}^{N_1-1}\left[u(n_1,k_2)\right] W_{N_1}^{n_1k_1} \qquad (65)$$

This is now a true two-dimensional DFT with the mapping of n and k given by:

$$n = (N_2\, n_1 + N_1\, n_2) \bmod N \quad (66a)$$

$$k = \left([(N_2^{-1}) \bmod N_1]\, N_2 k_1 + [(N_1^{-1}) \bmod N_2]\, N_1 k_2 \right) \bmod N \quad (66b)$$

In this example the mapping of n is of the simplest form and that of k is the so called Chinese Remainder Theorem (CRT).

Having mapped the original sequence x(n) into the two dimensional array $y(n_1, n_2)$, equation (65) indicates that the desired DFT can be evaluated by the following two steps:

(i) Performing an N_2-point DFT on each row of matrix $y(n_1, n_2)$. This corresponds to a total of N_1 N_2-point row DFT's and would convert the matrix $y(n_1, n_2)$ into the matrix $u(n_1, k_2)$ as shown in figure ???.

(ii) Performing an N_1-point DFT on each column of the resultant matrix, $u(n_1, k_2)$, to yield $Y(k_1, k_2)$. The desired output sequence X(k) is related to $Y(k_1, k_2)$ via the mapping given by (66b).

Example :

In this example we consider the evaluation of the DFT of a 35-element data array x(n) (n=0 to 34) via the mapping techniques discussed so far. Let us take $N=N_1 \times N_2=7 \times 5=35$ i.e. $N_1=7$ and $N_2=5$. The data array x(n) is first mapped into a two dimensional array $y(n_1, n_2)$ via the mapping given by equation (66a) i.e. $n = (5n_1 + 7n_2) \bmod 35$ (67)

The array $y(n_1, n_2)$ would thus be as follows:

$$y(n_1,n_2) = \begin{bmatrix} x(0) & x(7) & x(14) & x(21) & x(28) \\ x(5) & x(12) & x(19) & x(26) & x(33) \\ x(10) & x(17) & x(24) & x(31) & x(3) \\ x(15) & x(22) & x(29) & x(1) & x(8) \\ x(20) & x(27) & x(34) & x(6) & x(13) \\ x(25) & x(32) & x(4) & x(11) & x(18) \\ x(30) & x(2) & x(9) & x(16) & x(23) \end{bmatrix}$$

The next step is to perform the DFT of each row of this matrix. Obviously in this example this involves seven 5-point DFT's as shown in Figure 14. The result of these row DFT's is a new matrix denoted by $u(n_1,k_2)$.

Next the DFT of each column of the matrix u is evaluated. As shown in Figure14 this involves five 7-point DFT's and yields the matrix $Y(k_1, k_2)$. The two dimensional array $Y(k_1, k_2)$ contains desired DFT results X(k), with the allocations governed by the mapping given by equation (66b) i.e. $X(k)=Y(k_1,k_2)$ with

$$k = \left([(N_2^{-1}) \bmod N_1]\, N_2 k_1 + [(N_1^{-1}) \bmod N_2]\, N_1 k_2 \right) \bmod N$$

$$k = [(3)5k_1 + (3)7k_2] \bmod 35$$

$$= (15k_1 + 21k_2) \bmod 35$$

The array $Y(k_1, k_2)$ would therefore have the following arrangement:

$$Y(k_1,k_2) = \begin{bmatrix} X(0) & X(21) & X(7) & X(28) & X(14) \\ X(15) & X(1) & X(22) & X(8) & X(29) \\ X(30) & X(16) & X(2) & X(23) & X(9) \\ X(10) & X(31) & X(17) & X(3) & X(24) \\ X(25) & X(11) & X(32) & X(18) & X(4) \\ X(5) & X(26) & X(12) & X(33) & X(19) \\ X(20) & X(6) & X(27) & X(13) & X(34) \end{bmatrix}$$

In a practical implementation, the IMSA100 transversal filter can be used to perform these short row and column DFT's via the prime number transform algorithm described earlier. The important fact to note here is that each set of row (or column) DFT's consists of a number of totally independent short transforms (see Figure 14). This allows various degrees of parallelism to be exploited very easily in acheiving the required specification.

For example a single A100 based DFT processor can be used to sequentially perform all the row DFT's followed by the column DFT's, or when extremeley high processing speed is essential, several such DFT processors can be employed in parallel to complete the independent row (or column) DFT's. In the extreme case, it is possible to compute all row and column DFT's concurrently in a pipelined system arrangement. The INMOS concurrent processor family (transputers) when combined with the IMS A100(s) provide an ideal environment for exploiting these algorithms.

In arriving at equation (65) we applied the conditions given by (64) to equation (63). This resulted in the mapping given by (66) on which the last example was based . It is possible to use other values for α, β, γ, and δ than those given by (64).

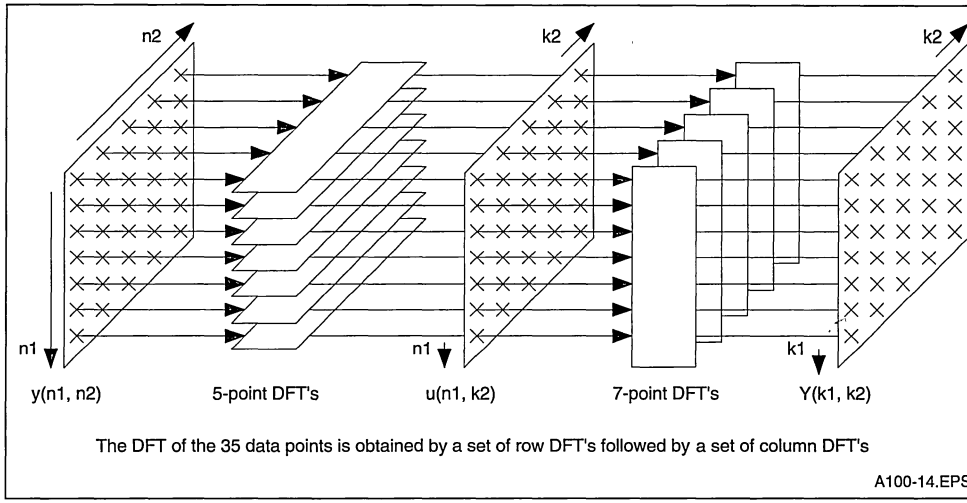
For example we could have used:

$$\gamma = \delta = 1$$

$$\alpha = (N_2^{-1}) \bmod N_1 \qquad (68)$$

$$\beta = (N_1^{-1}) \bmod N_2$$

Figure 14 : Schematic Representation of Equation 65 for N = N_1 x N_2 = 7 x 5



This would have resulted in the mappings for n and k to be interchanged i.e.

$$n = \left([(N_2^{-1}) \bmod N_1]\, N_2 n_1 + [(N_1^{-1}) \bmod N_2]\, N_1 n_2 \right) \bmod N \quad (69a)$$

$$k = (N_2 k_1 + N_1 k_2) \bmod N \quad (69b)$$

Another interesting possibility is

$$\alpha = \beta = \gamma = \delta = 1 \quad (70)$$

This would result in the simple mapping for both n and k i.e.

$$n = N_2 n_1 + N_1 n_2 \quad (71a)$$

$$k = N_2 k_1 + N_1 k_2 \quad (71b)$$

but requires a modification in the W. We can see this by substituting (70) into (63) which gives:

$$Y(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1,n_2)\, W_{N_2}^{N_1 n_2 k_2} \right] W_{N_1}^{N_2 n_1 k_1} \quad (72)$$

By defining

$$W'_{N_2} = W_{N_2}^{N_1} = e^{-2\pi j N_1/N_2}$$

and

$$W'_{N_1} = W_{N_1}^{N_2} = e^{-2\pi j N_2/N_1}$$

Equation (72) can be rewritten as:

$$Y(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1,n_2)\, W'^{n_2 k_2}_{N_2} \right] W'^{n_1 k_1}_{N_1} \quad (73)$$

This equation is very similar to (65), with the exception that a modified W is used. Equation (73) also maps into an arrangement similar to Figure 14. The DFT's of course, would have to be calculated with the modified W. This still can be done via the prime number transforms by simply replacing W with W′ in the transform.

V.3.b. CASE 2 — COMMON FACTOR DECOMPOSITION

Having covered the case where N_1 and N_2 were relatively prime, we now go back to equation (58) and consider the case where N_1 and N_2 have a common factor r, i.e.

$$(N_1,N_2)=r$$

Remember that we applied (59) to (58) which made the last term in (58) equal to unity i.e. we have :

$$Y(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1,n_2)\, W_{N_2}^{\beta L_2 n_2 k_2} \right] W_N^{M_1 L_2 n_1 k_2}\, W_{N_1}^{M_1 \gamma n_1 k_1} \quad (74)$$

Unlike the previous case we cannot use equation (60) to make the second terms in (74) equal to unity. This is because the equations in (60) would violate the necessary requirement, specified by (51), for one-to-one mapping.

The term $W_N^{M_1 L_2 n_1 k_2}$ is referred to as a 'twiddle factor'. Referring to (51) and remembering that we have already used the conditions given by (59), we can set

$$M_1 = L_2 = \beta = \gamma = 1 \quad (75)$$

which gives

$$Y(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} y(n_1,n_2)\, W_{N_2}^{n_2 k_2} \right] W_{N_1}^{n_1 k_2}\, W_{N_1}^{n_1 k_1} \quad (76)$$

with the mapping given by:

$$n = n_1 + N_1 n_2 \quad (77a)$$
$$k = N_2 k_1 + k_2 \quad (77b)$$

Note that equation (76) is very similar to equation (65) with the exception of the existence of the twiddle term. Equation (76) can be interpreted as shown in Figure 15. It can be seen that when N_1 and N_2 have a common divisor, N-complex multiplications have to be performed between the row and column DFT operations. In the previous case where N_1 and N_2 were assumed to be relatively prime no such multiplications were needed making the former mapping more efficient and easier to implement.

V.4. Extension to multiple dimensions

The concepts presented in this application note were concentrated around a two-dimensional mapping. There is no reason why the same concepts cannot be extended to more dimensions. For example if $N=N_1 \times N_2 \times N_3$

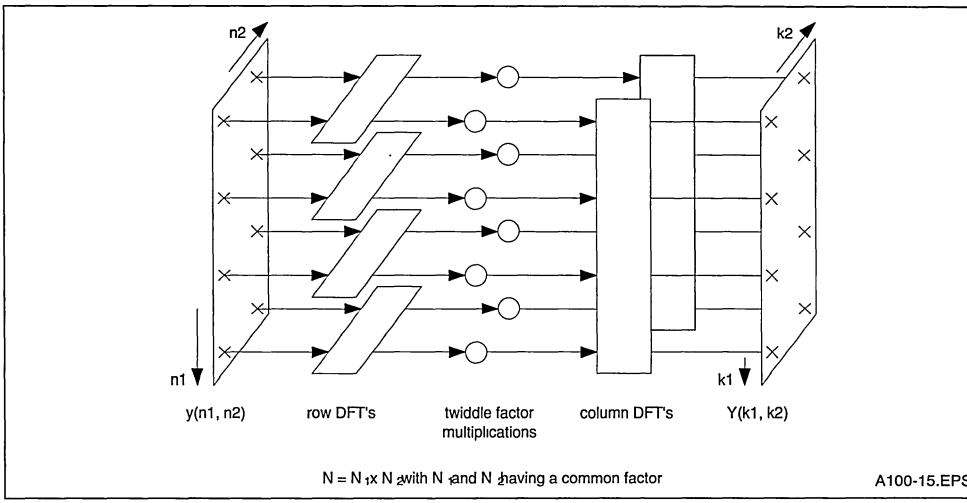
where N_1, N_2 and N_3 are relatively prime. The original N-point transform can be carried out via $N_2 \times N_3$, N_1-point, transforms followed by $N_1 \times N_3$, N_2-point, transforms followed by $N_1 \times N_2$, N_3-point, transforms. The easiest way to see this is to first map the N-point transform into a two-dimensional one with dimensions

$$N^{1} = N_3' \text{ and } N_2' = N_1 N_2$$

This consists of N_3, $N_1 \times N_2$-point, transforms (row DFT's) followed by $N_1 \times N_2$, N_3-point, transforms (column DFT's). Each one of the $N_1 \times N_2$-point transforms can then be decomposed into N_1, N_2-point, transforms followed by N_2, N_1-point, transforms.

Note that these multidimensional index mappings apply to both prime factor and common factor decompositions. In fact radix-2 FFT is nothing more than a common factor decomposition where all the factors N_1, N_2, N_3, N_4..... are made equal to 2. The advantage of the prime factor over the common factor decomposition is in that no twiddle matrix multiplications are needed for the prime

Figure 15 : Mapping of an N point DFT inti Dimensions



VI. REFERENCES

1 An algorithm for the machine calculation of complex Fourier series, Cooley J.W., Tukey J.W.,
Math. Comput., Vol.19, pp297-301, April 1965

2 On computing the discrete Fourier transform, Winograd S.,
Proc. Nat. Acad. Sci. USA, Vol.73, No. 4, pp. 1005-1006, April 1976

3 Discrete Fourier transforms when the number of data samples is prime, Rader C.M.,
Proc. IEEE, Vol.56, pp 1107-1109, June 1968

4 The chip z-transform algorithm and its application, Rabiner L.R., et al,
The Bell System Technical Journal, May-June 1969, pp 1249-1292

5 Index mappings for multidimensional formulation of the DFT an convolution, Burrus C.S.,
IEE Trans. on ASSP, Vol.25, June 1977, pp 239-242

6 DFT/FFT and convolutional algorithms-theory and implementation, Burrus C.S., Parks T.W.,
Wiley-interscience Publication, New York 1985

7 Number theory in digital signal processing, McClellan J.H., Rader C.M.,
Englewood Cliffs, NJ, Prentice-Hall Inc, 1979

8 The relationship between two fast Fourier transforms, Good I.J.,
IEE trans. on Comput., Vol. C20, pp310-317, March 1971

CORRELATION AND CONVOLUTION WITH THE IMSA100

SUMMARY

I. INTRODUCTION

The correlation process is widely used in many electronic systems including instrumentation, communication, medical ultrasonics, radar, sonar, control systems and other signal processing environments. The basic reasons for this widespread use can be attributed to the many useful characteristics exhibited by the correlation process. These properties include

- The ability to recover a desired signal masked by noise or other interferences. This is particularly useful in noisy environments that arise in communication, radar, sonar and ultrasonic applications.
- Delay estimation capability which is essential in many applications including range measurement in navigation systems, radar, sonar and also system identification.
- The ability to recognize a given pattern within a signal.
- The auto-correlation of a signal is closely related to the power spectrum which has resulted in the application of the correlation process to spectral analysis.
- The correlation process provides a good characterization of many signals and has therefore been used in many prediction and estimation algorithms.

Convolution is closely related to the correlation process. Mathematically convolution is what happens in the process of filtering. It will be shown in the next section that both these functions involve a large number of multiplications and additions. Up to now, for the time domain implementation of these processes, many systems have used multiply-accumulator devices. Because of their inherent concurrency, the numerical evaluations involved in the convolution and correlation functions can be performed in parallel. But due to the high cost, power consumption requirement, and size restriction many digital systems use only a single (or possibly two) multiply-accumulator(s). This has resulted in a processing bottle-neck in the time domain evaluation of these functions. For example using a 16-bit

multiply and accumulator chip available today it is possible, for a 32-point digital correlator, to achieve at best a sampling frequency of around 100 to 300kHz. This is further reduced as the number of correlation points increases. Additional complexities occur as some form of address generator has to be used to sequence the data and the reference coefficients through the multiply-accumulator chip.

The IMSA100 VLSI chip overcomes these problems by incorporating 32 multiply-accumulators on a single chip. The sampling speed of the IMSA100 ranges from 2.5MHz to 10MHz depending on the reference-waveform word-size. (4, 8, 12 or 16 bits). It is the true parallelism incorporated in the systolic structure of the IMSA100 that allows such speed increases. The architecture of the IMSA100 has been designed in such a way that large numbers of these chips can be cascaded to perform high precision correlations involving more than 32 points at full speed. Alternatively it is possible to use multidimensional index mapping to decompose a long correlation/convolution into a number of short ones which can then be carried out by usoing a single or a small number of devices.

By suitable allocation of the coefficients, the IMSA100 can be used to perform 3×3, 5×5, or larger two-dimensional image convolutions.

In this application note the concepts of correlation and convolution are first introduced followed by their IMSA100 implementation issues. Partitioning techniques for decomposing a long correlation/convolution into a number of short ones are then described. Next an example of a two-dimensional image convolution is given. Finally some application areas of correlation and convolution are summarised.

II. CORRELATION CONCEPTS

The correlation between two functions is a measure of their similarity. This is illustrated in Figure 1 where three extreme cases are depicted.

Figure 1 : Illustration of Correlation Process

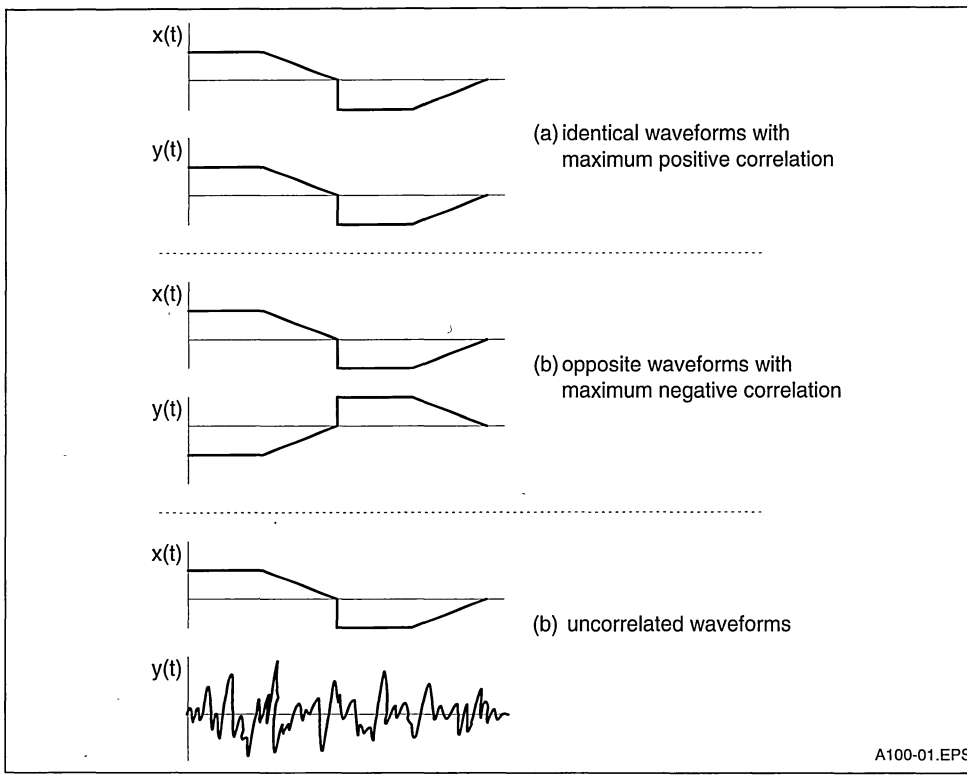


Figure 1a shows two waveforms which are absolutely identical and they have maximum positive correlation. The two waveforms in Figure 1b are similar, except for their polarities and as such they have maximum negative correlation. Finally Figure 1c shows one of the waveforms of Figure 1a and a noise like signal. As these two waveforms are completely dissimilar the correlation between them is expected to be very small or even zero.

Mathematically the correlation function between two waveforms x(t) and y(t) is expressed as

$$R_{xy}(\tau) = \lim_{T \to \infty} \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{+T}{2}} x(t)\, y(t+\tau)\, dt \quad (1)$$

$R_{xy}(\tau)$ is also referred to as cross-correlation between the two waveforms. For identical waveforms (ie correlating a waveform x(t) with itself) the correlation function is denoted by $R_{xx}(\tau)$ and is called the auto-correlation function.

Equation (1) can be interpreted as follows:

The cross-correlation function, $R_{xy}(\tau)$ between the two waveforms x(t) and y(t) is obtained by shifting one of the two signals in time by an amount equal to τ (i.e. modifying y(t) to y(t+τ)), multiplying the shifted waveforms by the other signal and integrating the product.

If the waveforms are periodic with a period T_0, equation (1) can be modified to:

$$R_{xy}(\tau) = \frac{1}{T_0} \int_{\frac{-T_0}{2}}^{\frac{+T_0}{2}} x(t)\, y(t+\tau)\, dt \quad (2a)$$

i.e. the integration is evaluated only over one period of the signal.

Figure 2 provides a graphical illustration of the process of cross correlation between two waveforms x(t) (Figure 2a) and y(t) Figure 2b. We start by multiplying the two waveforms and integrating over the interval $\frac{-T_0}{2} \leq \frac{+T_0}{2}$ yielding $R_{xy}(0)$. With $\tau = 0$, (ie no shift) it can be seen from Figures 2a & 2b that there is no overlap between non-zero parts of the two waveforms resulting in $R_{xy}(0)=0$ as shown in Figure 2f. To evaluate $R_{xy}(\tau)$, the waveform y(t) is left shifted by an amount equal to τ, giving y(t+τ), and the multiplication and integration is repeated. As the waveform y(t) is shifted to the left, there will be no overlap between non-zero parts of y(t+τ) and x(t) untill τ τ_1, see Figure 2c. At $\tau=\tau_1$, the non-zero parts of the two waveforms just begin to overlap. Figure 2d shows the position of y(t) when it is shifted by $\tau=\tau_2$. Here the non-zero parts of x(t) and y(t+τ_2) have have overlapped and the integration of the product of the two waveforms therefore yields a non-zero value for $R_{xy}(\tau_2)$ as shown in Figure 2f. As y(t) is shifted further the non-zero overlapping section of the two waveforms and hence the value of $R_{xy}(\tau)$ increase. When y(t) is shifted to the position shown in Figure 2e, full overlap occurs and $R_{xy}(\tau)$ will attain its maximum value of $R_{xy}(\tau_3)$ as shown in Figure 2f. Shifting y(t) further causes the value of $R_{xy}(\tau)$ to decrease as the two waveforms pass each other. Figure 2f shows the complete cross-correlation function between the two waveforms. You can confirm the shape of this cross-correlation function by evaluating equation (2a).

One interesting point to note here is that the maximum value of $R_{xy}(\tau)$ occurs at $t=\tau_3$ which is equal to the time-lag, T_L, between the two waveforms x(t) and y(t). This is how the correlation process is used to measure delays.

From figure 2 it can be seen that the cross correlation function could have been evaluated by shifting x(t) to the right instead of left shifting y(t). Mathematically this can be confirmed by defining a new variable t'=t+τ and substituting in (2a) which gives:

$$R_{xy}(\tau) = \frac{1}{T_0} \int_{\frac{-T_0}{2}}^{\frac{+T_0}{2}} x(t' - \tau)\, y(t)\, dt \quad (2b)$$

So far we have dealt with the correlation of analogue signals. For digital processing both waveforms x(t) and y(t) have to be sampled and digitalized. For discrete-time signals the process of correlation can be expressed as:

$$R_{xy}(mT) = \frac{1}{N} \sum_{k=0}^{N-1} x(kT)\, y[(k+m)\, T] \quad (3a)$$

At time t=kT equation (3a) requires future samples of y(t). Similar to the analogue case, (equation 2b), the above equation can be modified so that it only uses past samples of y(t), i.e.

$$R_{xy}(mT) = \frac{1}{N} \sum_{k=0}^{N-1} x[(k-m)T]\, y(kT) \quad (3b)$$

In equation (3b), T, donates the sampling period and should be chosen to ensure that the sampling rate is greater than twice the signals bandwidth (Nyquist rate). For the sake of simplicity the factor, T, is usually dropped from the indices of equations (3a) & (3b), i.e.

$$R_{xy}(m) = \frac{1}{N} \sum_{k=0}^{N-1} x(k)\, y(k+m) \quad (4a)$$

and

$$R_{xy}(m) = \frac{1}{N} \sum_{k=0}^{N-1} x(k-m)\, y(k) \quad (4b)$$

Where k and m are used to index the samples and N is the number of correlation points involved. In practice the correlation size N will depend on the duration of the two functions, and on their periodicity if they are periodic. From equations (4a) or (4b) it can be observed that direct evaluation of M samples of the cross-correlation function, R_{xy}, will involve M× N multiply-and-accumulate operations.

III. CONVOLUTION CONCEPTS

The convolution function is closely related to that of correlation. The convolution of two signals x(t) and y(t) is mathematically defined by:

$$C_{xy}(\tau) = \lim_{T \to \infty} \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{+T}{2}} x(t)\, y(\tau - t)\, dt \quad (5)$$

This equation is very much similar to equation (1) defining the correlation process. Their difference is that in convolution the signal y(t) is first time-reversed (i.e. is mirrored around t=0) and then shifted by τ. This time-reversed and shifted signal is then multiplied by x(t) and the product is integrated over all t's. Figure 3 graphically illustrates the process of convolution.

The process of convolution occurs in filters where the output of a filter is in fact the convolution of the input function, d(t), and the impulse response, h(t), of the filter (see the application note entitled 'Digital Filtering with the IMSA100'):

$$f(\tau) = \int_{-\infty}^{+\infty} d(t)\, h(\tau - t)\, dt \quad (6)$$

where f(τ) is the filter output.

For discrete-time signals equation (5) becomes

$$C_{xy}(m) = \frac{1}{N} \sum_{k=0}^{N-1} x(k)\, y(m-k) \quad (7)$$

which defines the convolution function for digital signals. Notice that like correlation, convolution involves carrying out N multiply-and-accumulations for each sample of $C_{xy}(m)$. Due to the high degree of similarity between correlation and convolution functions, the same hardware can be used to perform both functions. All that needs to be done is to time-reverse one of the waveforms for the convolution process.

Figure 2 : Graphical Illustration of Correlation Process

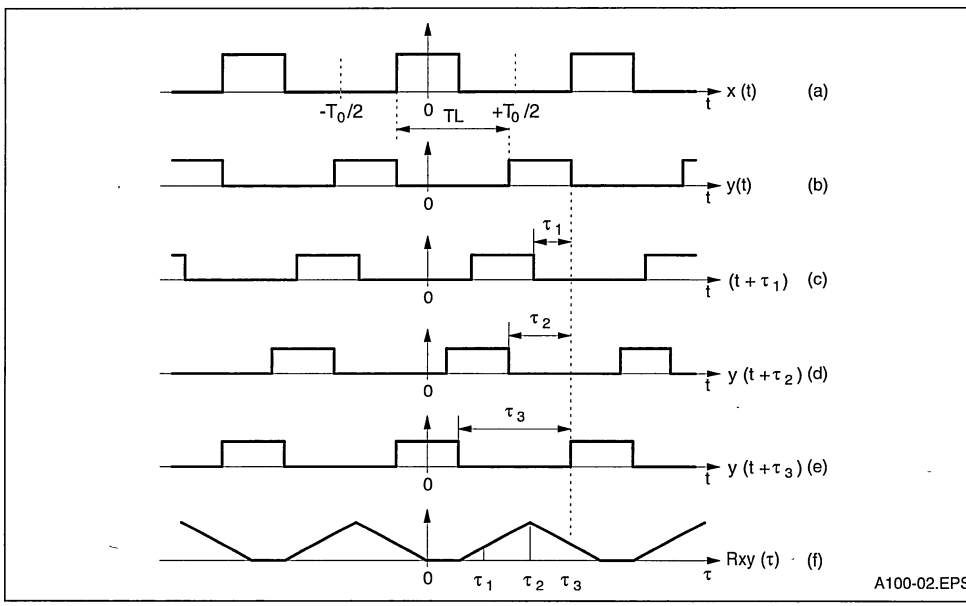
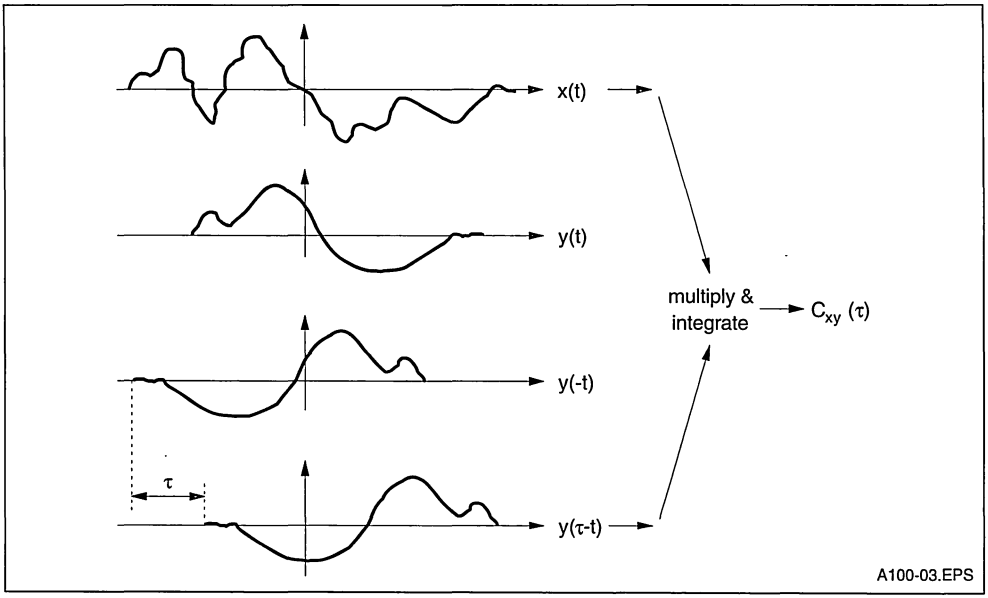


Figure 3 : Illustrating of Convolution Process



The following two sections deal with hardware implementations for the correlation and convolution functions. The first section deals with the conventional approach involving multiply-and-accumulator chips and points out the processing bottle-necks associated with these solutions. The second section shows how the IMSA100 signal processor can be configured to perform these functions efficiently and simply, at speeds not economically feasible with the conventional approach.

IV. CONVENTIONAL HARDWARE FOR TIME-DOMAIN EVALUATION OF CORRELATION

As discussed earlier, the processes of correlation and convolution are based on multiplying a delayed version of a sequence of samples by another sequence and summing the products. Conceptually this could be mechanised, as shown in Figure 4, by providing two shift registers to hold all the values of x's and y's required for the computation, a further shift register to provide the delay (mT), an array of multipliers for forming the products, and a multi-input adder for the final summation. In the example of Figure 4 the output would correspond to $R_{xy}(2)$, as a delay of two stages is incorporated in the path of signal x(kT) giving x((k-2)T) (see equation 3b).

Up to now, due to the large number of multipliers and adders involved, it has not been possible to economically implement high precision correlators directly in the form given by Figure 4. Instead to minimize the size, cost and power consumption, a single multiply-and-accumulator is usually used and time-shared between all the multiplications. Figure 5 shows a schematic block diagram of a conventional correlator implementation. The system consists of memories to hold samples of the two signals to be correlated, a multiply-accumulator and an address generator hardware which is responsible for sequencing the correct order of signal samples through the multiply-and-accumulator. The obvious disadvantage of this arrangement is the processing bottle-neck caused by using a single multiply-and-accumulator to sequentially evaluate what is inherently a concurrent problem. Assuming a multiply-accumulate time of T_{mac}, for an N-point correlator implemented using a single multiply-accumulator, the maximum sampling rate would be

$$f_{max_s} = \frac{1}{NT_{mac}} \quad (8)$$

For example if T_{mac}= 100ns and N=100, then a signal sampling frequency of at most 100kHz would be possible.

Figure 4 : Schematic Diagram for an Ideal Correlator Hardware

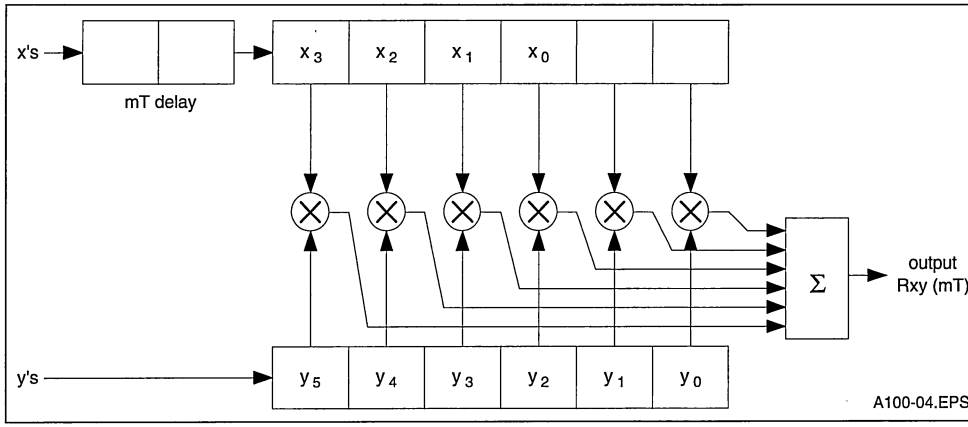
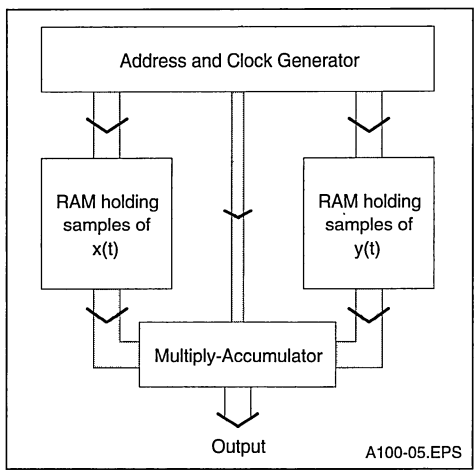


Figure 5 : Block Diagram of a Conventional/convolver



Many applications such as radar and communication require faster processing rates than can be achieved using a single multiply-and-accumulator. (Some improvements can be achieved by carrying out the processing in the frequency domain at the cost of introducing some complexity. However here we are only concerned with the time-domain approach. A separate application note entitled 'Discrete Fourier Transform with the IMSA100' deals with the time-domain to frequency-domain transformations)

In applications where a fast processing rate is essential, a trade-off is often made between the correlator precision and its speed. For example if one or both of the signals x and y are assumed binary, the multiplications become simple binary AND operations, and it would be possible to implement a high speed low precision correlator. In fact many correlator chips available today are of this type and have very low precision compared to those implemented from multiply-accumulators.

The IMSA100 chip on the other hand is the first high-precision high-speed VLSI implementation of a single-chip correlator. It provides a numerical accuracy in excess of that of the 16-bit multiply and accumulators while allowing sampling rates in the MHz region. The next section illustrates how this chip can be used to perform fast and highly accurate correlation and convolution functions.

V. THE IMSA100 IMPLEMENTATION OF CORRELATION/CONVOLUTION

The IMSA100 is a 32-stage correlator (convolver) in which the samples of the two signals to be correlated can be represented as up to 16-bit words. This corresponds to a signal dynamic range of 96 dB's. A number of these devices can also be cascaded, without the need for any external components, to provide much longer correlators while preserving a high degree of accuracy. The IMSA100 chip (or cascaded chips) can be fully memory mapped and used as a peripheral accelerator to a host processor.

To understand the architecture of the IMSA100 let us first consider the basic function of a simple 3-point correlator shown in Figure 6a. The three

samples of the first signal (reference signal) i.e. x_0, x_1 and x_2 are loaded in three registers feeding an array of three multipliers. The samples of the second signal i.e. y_0, y_1, y_2,.... are fed into a 3-stage shift register whose outputs are also connected to the multipliers. A three input adder combines the products to give the correlation function. As the samples of the signal y are shifted through the shift register, the output of this hypothetical correlator (assuming the shift register is reset at the start) will be:

$$y_0 x_2 \, , \, y_0 x_1 + y_1 x_2 \, , \, y_0 x_0 + y_1 x_1 + y_2 x_2 \, , \, y_1 x_0 + y_2 x_1 + y_3 x_2, \ldots$$

The correlator structure in Figure 6a can be modified to that given in Figure 6b without affecting the functionality. In Figure 6b the multi-input summation process is avoided and replaced by a chain of delay-and-add units. The input, supplying the signal y, is also made common to all of the multipliers. Note also that the signal samples x_0, x_1, x_2 are stored in the opposite direction to that of Figure 6a. Supplying the input sequence of samples y_0, y_1, y_2, y_3,..... to the structure of Figure 6b and simultaneously shifting the partial products along the delay-and-add chain, it is straightforward to confirm that the output sequence would be

$$y_0 x_2 \, , \, y_0 x_1 + y_1 x_2 \, , \, y_0 x_0 + y_1 x_1 + y_2 x_2 \, , \, y_1 x_0 + y_2 x_1 + y_3 x_2 \, , \ldots$$

This sequence is absolutely identical to that obtained from Figure 6a. In other words the structure in Figures 6a & 6b have identical functionality and both can be used to perform correlation between two sequences. The IMSA100 architecture is based around this modified structure. The major processing part of the chip incorporates 32 multipliers and a 32-stage delay-and-add chain as shown in Figure 7.

At this point the interested reader is advised to consult the data sheet of the IMSA100 for full details.

In order to correlate two sequences x(k) and y(n), the samples of one of the two signals, say x(k)'s, should be stored in one set of the IMSA100's coefficient registers. These samples should be loaded from left to right with the last sample of x(k) stored in the coefficient register associated with the last multiplier. If the reference waveforms x(k) is less than 32-samples long, any unused left-most coefficient registers should be set to zero. For a 30-sample reference signal, this allocation is depicted in Figure 8.

Figure 6 : Relating the IMSA100 architecture to that of a correlator

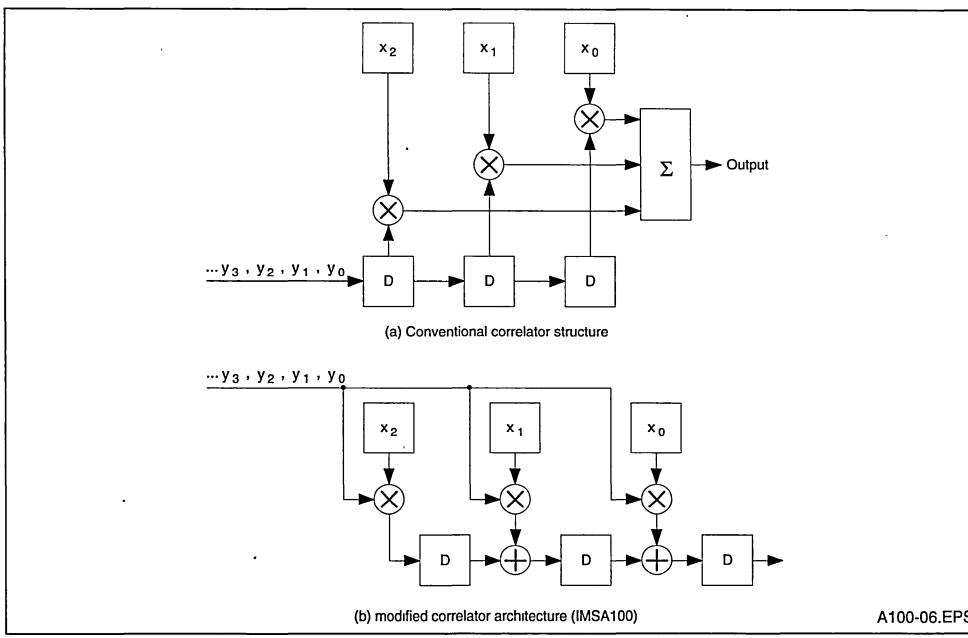


Figure 7 : User's Model of the IMSA100

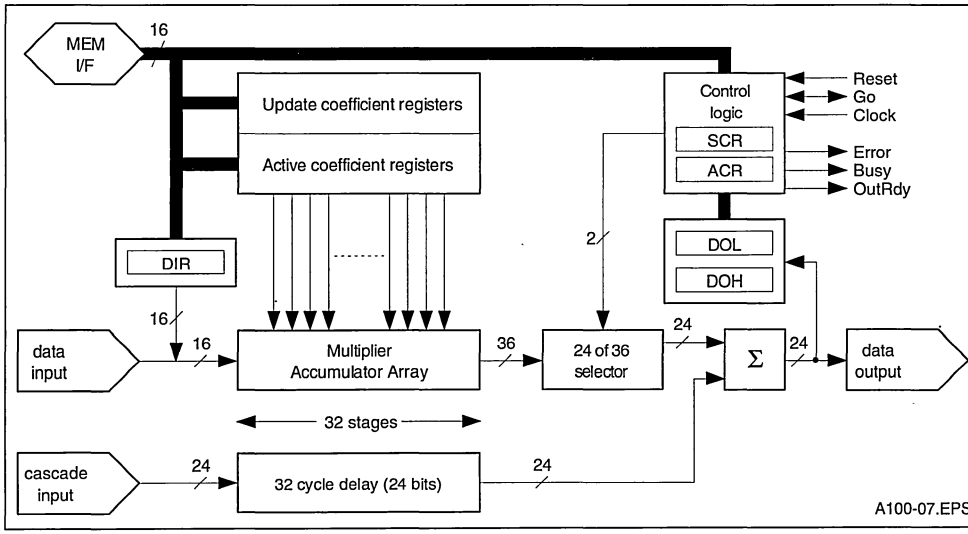
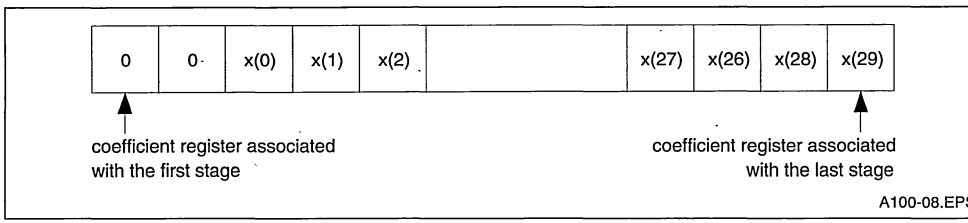
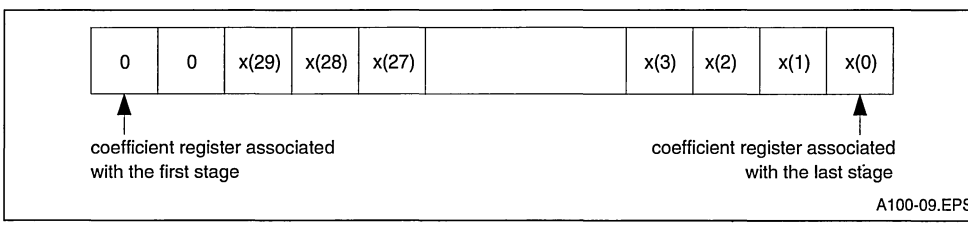


Figure 8 : Example of the reference signal allocation for a 30-point correlator



The samples of the other signal y(n) are then applied to the input of the IMSA100. As shown earlier the output sequence would correspond to the cross-correlation function of the two signals. If the two signals x(k) and y(n) are to be convolved rather than correlated, the reference signal x(k) should be loaded in the coefficient registers in the opposite direction. The register allocation for a 30-point convolver is shown in Figure 9. As discussed in the data sheet, the IMSA100 processor has two sets of coefficient registers. At any instant in time one set of coefficients is applied to the multiplier array, whilst the other set can be accessed via the IMSA100 memory interface.

Figure 9 : Coefficient register allocation for a 30-point correlator



For correlations (convolutions) dealing with real signals, one set of these coefficients would be sufficient. The second set can be used to hold a different reference signal and if necessary the function of the two memory banks can be interchanged by performing a write operation to the 'Bank swap' bit of a control register. Such an operation would initiate the correlation (convolution) of the input signal with the second reference waveform. The existance of the two coefficient register sets and the continuous bank-swap mode allows the IMSA100 to perform complex (correlation)convolution, where both the reference and the input signal have real and imaginary components. This configuration is discussed in the application note 'Complex Processing with the IMSA100'.

For the IMSA100 the data-word length is 16 bits whilst the coefficient-word length can be programmed to be 4, 8, 12 or 16 bits. The maximum data throughput (the sampling rate) is a function of the coefficient size. Table 1 relates the coefficient size to the maximum sampling rate and indicates the effective number of multiply-and-accumulate operations per second in each case. The last column shows the effective number of multiply-and-accumulates when four devices are cascaded.

Table 1

Coefficient word size (bits)	Sampling rate (MHz)	Effective number of multiply-and-accumulates (Millions/second)	
		Single A100	Four A100s
4	10	320	1280
8	5	160	640
12	3.3	106	424
16	2.5	80	320

The strength of the IMSA100 can be appreciated by comparing the effective number of multiply-and-accumulations/sec with that of multiply-accumulator IC's which range from 5--10 million/sec.

As discussed in the A100 data sheet, in order to preserve complete numerical accuracy no truncation or rounding is carried out on the partial products in the multiply-and-accumulation array. The output is thus calculated to 36 bit precision which ensures no overflows. A barrel-shifter at the output of the multiply-and-accumulate array allows 24 bits from these 36 bits to be selected (sign-extended if necessary) and rounded for output. This selection can be programmed via a control register. The programming details can be found in the IMSA100 data sheet.

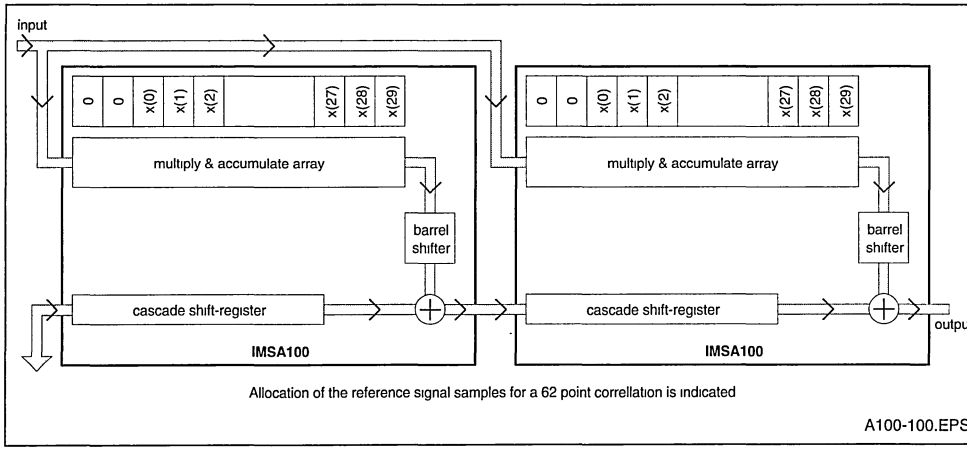
The architecture of the IMSA100 has been designed to allow large numbers of these devices to be cascaded for correlations (convolutions) involving more than 32-stages, the devices can be cascaded while preserving a high degree of accuracy and without the need for any external components. This is made possible by incorporating on chip a 32 stage, 24-bit wide, shift register and a 24-bit adder which combines the output of the barrel-shifter with that of the 32-stage shift register (see Figure 7).

The IMSA100 chips can be cascaded by simply connecting the output of the each device to the cascade input of the following device. The input is common to all cascaded devices. The effect of such an arrangement is that the output of the first device is delayed by 32 cycles, before being added to that of the next device. Figure 10 illustrates how, for example, a 64-point correlator can be implemented using two IMSA100 devices. The allocation of the reference signal samples is also indicated in this illustration. In this arrangement the barrel-shifter in each device acts as a data scalar (with rounding). The cascading process can be considered as a block-floating point operation where the common exponent is determined by the extent of the shift carried out by the barrel-shifter. With this cascading technique a very high degree of accuracy is preserved because the output scaling is only performed after every 32-multiply and accumulation stages and not at any intermediate stage.

For convolution purposes the reference signal should be loaded into the coefficient stage in the opposite direction to that shown in Figure 10.

A very important feature of the IMSA100 transversal filter is that the part is fully memory mappable. Apart from the two coefficient memory banks, which can be accessed via the IMSA100 standard memory interface, the input and output of the device are also accessible from the same interface. This feature allows the part to be used either with its input and output data communicated through the dedicated ports or through the memory interface. The latter options allows the device to be easily interfaced to a host processor and used as a high speed peripheral. The status and control registers of the IMSA100, accessible via the memory interface, provide full control of the part by the host processor.The memory interface can also be used as a facility for system diagnostics, as the host processor can act as a watch-dog in systems involving arrays of IMSA100's. Full specification of the IMSA100, its status and control registers and its standard memory interface are detailed in the data sheet available from SGS-THOMSON.

Figure 10 : Cascading two IMSA100 devices to obtain a 64 point correlator



Allocation of the reference signal samples for a 62 point correllation is indicated

A100-100.EPS

VI. DECOMPOSITION OF LONG CORRELATIONS AND CONVOLUTIONS

A single IMSA100 device is effectively a 32-tap correlator (convolver) in which the samples of the two signals to be correlated can be expressed in upto 16-bit words. As described earlier, one method to deal with correlations/convolutions involving more than 32 points is to use several cascaded devices to achieve a longer correlator/convolver. For such an arrangement, and with 16-bit coefficients, tha data rate can be as high as 2.5 Million samples/sec.

Alternatively it is possible to use various decomposition techniques to partition a long correlation/convolution into a number of shorter ones, which can then be carried out by a single or a small number of IMSA100 devices. The host machine would merely combine the results from these short correlations/convolutions to obtain the overall result. The advantage of this approach, compared to using single MAC based processors, is a significant reduction in the required memory bandwidth. This is why even a medium-speed general purpose microprocessor can achieve a very high performance when combined with the IMSA100.

A simple way to decompose a long correlation/convolution of length N, between waveforms x and y, is to break up one of the waveforms, say x, into consecutive blocks of 32 sample. Each one of these blocks can then be correlated/convolved with the whole of the waveform y by loading each block into the IMSA100 coefficient registers, and using y as the input sequence. The output from these correlations/convolutions can then be combined by displacing each partial result by 32 samples, with respect to the previous one, and performing an addition operation. Note that the coefficient registers, containing blocks of waveform x, need only be updated once every time the whole of the waveform y is fed through the device, resulting in a significant saving in the memory bandwidth. The block size of 32, suggested above, whould mean that a single IMSA100 would be sufficient. However processing speed can be improved by using cascading devices to perform these partial correlations/convolutions. With suitable memory mappings, hosts such as SGS-THOMSON transputers can use their on-chip DMA engine to feed the IMSA100 devices with the samples of the waveform y.

A more complicated decomposition technique, to be described here, is based on the multidimensional index mappings (references 1 & 2). These techniques are applicable to cyclic convolutions/correlations. However all convolutions/correlations can be made cyclic by adding zero terms to the end of the data blocks. As an example, consider the following cyclic correlation:

$$C(k) = \sum_{n=0}^{N-1} x(k+n)\, y(n) \quad (9)$$

where the indices are evaluated modulo N. The arrays C, x, and y can be mapped into multidimensional arrays C', x', and y', the requirement being that the mapping should be one-to-one and cyclic

in at least one dimension. The map, in general, can assume many different forms, but the one particularly useful is the linear form. For a simple two-dimensional decomposition such a map would be of the form:

$$n = (M_1 n_1 + M_2 n_2) \bmod N \quad (10)$$

Note that n is evaluated modulo N, making the map cyclic in n. In order for this map to be unique and one-to-one, the mapping constants M_1 and M_2 must satisfy certain conditions. These conditions are summarised in section 6 of the IMSA100 Application Note 2 which is available from INMOS and will not be repeated here.

As an example let us map the arrays in equation (9) into two-dimensional matrices of dimensions N_1 and N_2 where $N=N_1 \times N_2$, we can use the mapping given by equations (10) for n and k giving

$$C(M_1 k_1 + M_2 k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2}^{N_2-1} x(M_1 k_1 + M_2 k_2 + M_1 n_1 + M_2 n_2)\, y(M_1 n_1 + M_2 n_2)$$

or

$$C'(k_1, k_2) = \sum_{n_1}^{N_1} \sum_{n_2}^{N_2} x'(k_1 + n_1, k_2 + n_2)\, y'(n_1, n_2) \quad (12)$$

This is now a true two-dimensional convolution which can be made cyclic along n_1 if M_1 is made a multiple of N_2, and/or cyclic along n_2 if M_2 is made a multiple of N_1. With these conditions, inspection of equation (12) shows that the long N-point circular correlation can be performed by N_1^2, N_2-point correlations or N_2^2, N_1-point correlations. This involves correlating each row (or column) of the matrix y' with all the rows (or columns) of the matrix x'. These short circular correlations can be efficiently performed by the IMSA100, with the host merely adding partial results. The approach is particularly efficient as it is possible to load one row (or column) of the matrix y' into the coefficient memory of the device and to feed all the rows (or columns) of the matrix x' successively to the input of the device to obtain partial results, for the elements in the matrix C'. The fact that with this algorithm, the coefficient memories need only be updated occasionally (once every time all the elements of the matrix x' are fed into the device) results in an impressive reduction in the memory bandwidth requirement. This is why, even with a general purpose microprocessor, as the host, very impressive perfomance can be achieved.

In the example given here, we concentrated around a two-dimensional mapping. It is important to realise that the same decomposition concepts can be extended to more dimensions. The easiest way to see this is to start with a two dimensional decomposition and then partition the rows of the two-dimensional matrices further. For example if

$$N = N_1 \times N_2 \times N_3$$

the original N-point correlation can be carried out via N_3^2, $N_1 \times N_2$-point correlations. However, each one of the $N_1 \times N_2$-point correlations can further be decomposed, as before into N_1^2, N_2-point correlations.

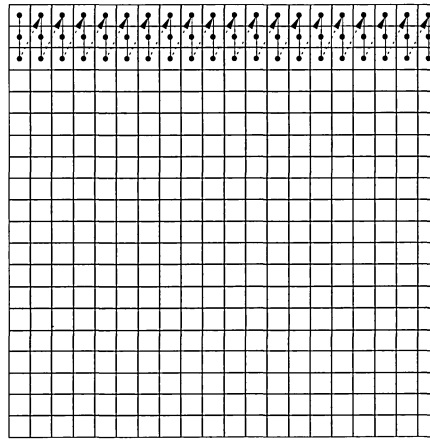
VII. 2-D IMAGE CONVOLUTIONS WITH THE IMSA100

Many applications including image processing require 2-D convolutions and correlations. Such operations are needed in image filtering, edge detection, etc. There are many ways that the IMSA100 can be used to speed up these operations. This section gives an example of how the device can be used to perform 3× 3, 5× 5, or larger convolutions.

Figure 11a shows a 20× 20-pixel image which is to be convolved with the 3× 3 reference matrix given by Figure 11b. One way to achieve this is to load the reference matrix, as shown in Figure 11c, in one of the IMSA100 coefficient register banks, and sequence the image data through the device as shown by the arrowed path in Figure 11a. In this way every third output sample of the IMSA100 would correspond to a valid filtered pixel for the second row of the image. To proceed, the same sequence, moved down by one row, is then passed through the device which provides the filtered results for the nest row and so on. A single IMSA100 can deal with reference matrices as big as 5×5.

An alternative arrangement which gives a better throughput is one where, as shown in Figure 12a, 7 zeroes are inserted in the IMSA100 coefficient registers (between terms corresponding to the columns of the reference matrix). The data sequencing would be as shown in Figure 12c, where ten pixels from a given column are fed through the device before moving to the next column. In this scheme the first nine rows of the image are filtered in one scan, with 80% of the output data samples being valid. (Note that, using a single device, the number of inserted zeroes can be increased from 7 to 11, allowing 13 image rows to be filtered in each scan.)

Figure 11 : Example of a 3 x 3 image convolution/correlation with the IMSA100



(a) 20 x 20 pixel image, arrows show the required data sequencing

a	d	g
b	e	h
c	f	i

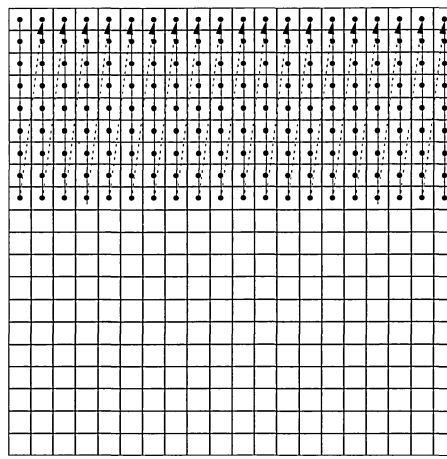
(b) 3 x 3 convolution matrix

0	0	0	0	.....		0	0	a	b	c	d	e	f	g	h	i

(c) Coefficient register allocation for the 3 x 3 convolution

A100-11.EPS

Figure 12 : Improved version of the 3 x 3 image convolution/correlation with the IMSA100



(a) 20 x 20 pixel image, arrows show the required data sequencing

a	d	g
b	e	h
c	f	i

(b) 3 x 3 convolution matrix

9 zeroes | 7 zeroes | 7 zeroes

.....		a	b	c	0	.....		0	d	e	f	0	.....		0	g	h	i

(c) Coefficient register allocation for the 3 x 3 convolution

A100-12.EPS

The examples given here are just a small subset of possible arrangements. Remembering that the IMSA100 devices can be cascaded or used in parallel, numerous other implementations for image processing become possible.

VIII. SOME APPLICATION EXAMPLES OF CORRELATION AND CONVOLUTION

Correlation and convolution are encountered in numerous applications of digital signal processing, this section summarises some of the application areas where these techniques are used.

VIII.1 Delay and periodicity estimation

The correlation process can be used to estimate the time delay between two similar signals. Figure 13 shows two signals x(t) and y(t) which are identical in shape but have a time delay between them. If these two signals are correlated the cross-correlation function would attain a maximum when y(t) is delayed by an amount equal to the delay between the two waveforms. This is illustrated in Figure 13c where the peak of the cross-correlation function occurs at $t=T_d$ where T_d is the delay between the two waveforms. This technique has applications in areas such as radar, sonar and medical ultrasonics where a measurement of the time delay between the transmitted signal and the return echo from an object gives an indication of the range of that object.

The same technique can also be used to measure the period of a repetitive signal. This can be achieved by correlating the signal with itself i.e. by calculating its auto-correlation function as illustrated in Figure 14 the auto-correlation of a periodic signal exhibits peaks, spaced a distance, T_0, apart where T_0 is the period of the signal. One application of this technique is pitch-period measurement in speech signals. The time gap between the peaks in the auto-correlation function of a segment of speech provides an estimate for the pitch period of voiced speech.

Figure 13 : Delay Estimation using Correlation Process

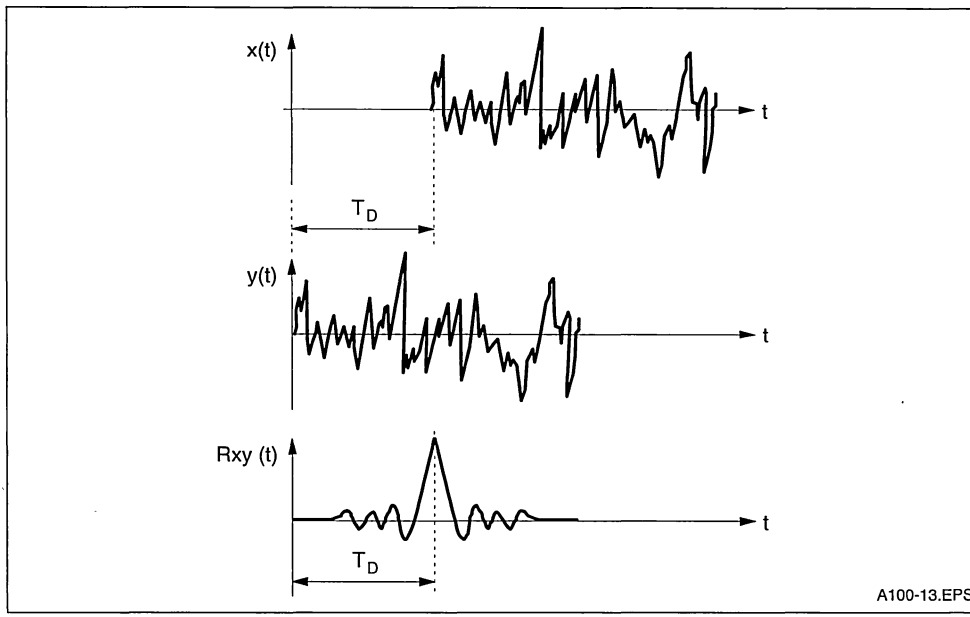
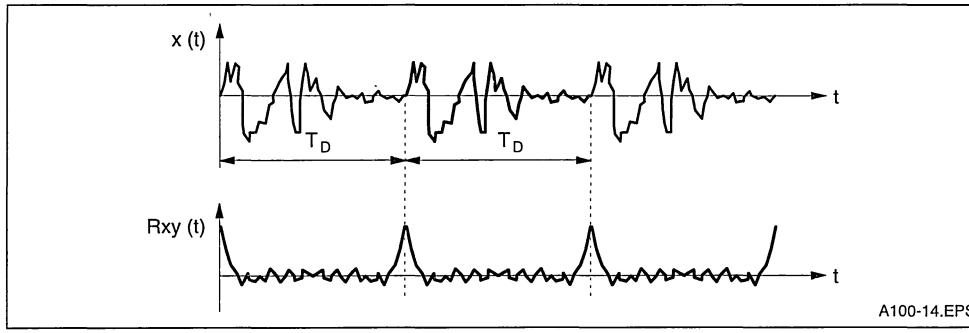


Figure 14 : Application of Correlation to the Periodicity Measurement



VIII.2 Noise reduction using correlation techniques

In many real-world applications the signals to be processed are immersed and possibly masked by noise. Such situations occur in noisy communication channels, long-range radar and sonar systems. In such cases correlation techniques can be used to extract and detect the signal from the background additive noise. This is achieved by correlating the noisy signal with a replica of the expected signal waveform. While the noise is uncorrelated with the replica signal, the signal immersed in noise will strongly correlate with the replica signal giving a large output value, well above the background noise. Mathematically this can be argued as follows (the proof here is not rigorous but does make the point):

Let the signal waveform consist of N samples values s_0, s_1, s_2,.....s_{N-1}. Suppose this signal is correlated by samples of a white noise having a standard deviation of σ_n (and variance σ_n^2). The ratio of the signal power to that of the noise prior to any processing is thus:

$$\frac{\text{Signal Power}}{\text{Noise Power}} = \frac{\sigma_s^2}{\sigma_n^2} \quad (13)$$

where σ_s^2 is the signal variance. Suppose we correlate this noisy signal with a replica of the original signal in an N-point correlator. At the instant when the signal waveform masked by the background noise is aligned with its replica in the correlator, the output attains its maximum. At this instant the amplitude of signal component at the output of the correlator would be

$$s_{out} = s_0^2 + s_1^2 + s_2^2 + s_3^2 + \ldots + s_{N-1}^2 \cong N\,\sigma_s^2$$

The corresponding output signal power would thus be : Output Signal Power = $N^2\,\sigma s^2$ (14)

The noise would also be modified by the operation of the correlator. In this case each output noise sample is equal to the sum of weighted input noise samples-the weighting coefficients being, of course, the samples of the reference signal. Hence each output noise sample is equal to the summation of N independent random numbers having standard deviations $s_0\sigma_n$, $s_1\sigma_n$, $s_2\sigma_n$, ..., $s_{N-1}\sigma_n$. Since variances are additive in this case, the variance of the output noise samples is therefore

$$\sigma_{nout}^2 = s_0^2\,\sigma^2 + s_{12}\,\sigma^2 + \ldots + s_{N-1}^2\,\sigma^2 = N\,s_n^2\,\sigma s^2 \quad (15)$$

The ratio of the output signal power to that of the output noise is thus

$$\left(\frac{S}{N}\right)_{OUT} = \frac{\text{Output Signal Power}}{\sigma_{nout}^2} = \frac{N^2\,\sigma_s^4}{N\,\sigma_n^2\,\sigma_s^2}$$

$$= N\,\frac{\sigma_s^2}{\sigma_n^2} = N\left(\frac{S}{N}\right)_{INPUT}$$

This indicates that the correlation process improves the signal to noise ratio by

$$C_G = 10\log_{10}(N)\ \text{dB's} \quad (17)$$

which is defined as the 'Correlator Gain'.

VIII.3 Pulse-compression

Another application of correlation is in radar and sonar systems where pulse compression techniques are used to improve range resolution of the systems. In many active sonar and pulsed radar systems, a short pulse is transmitted followed by a listening period that represents a 'look' in the range dimension. The two way propagation duration, i.e. the time it takes for the pulse to travel to a target and back gives an indication of the range of that target. The range resolution i.e. the shortest distance between two targets that can be resolved, is equal to $\frac{\tau c}{2}$ where τ is the pulse duration and c is

the speed of wave propagation in the medium. For example for a radar system a 10μ s pulse corresponds to a range resolution of 1.5km. Better range resolutions necessitate a shorter pulse. Unfortunately the transmitted pulses cannot be made too short. This is because most systems are peak-power limited and a shorter pulse means less signal power which in turn can severely limit operational range of the system.

Pulse compression techniques allow a radar or sonar to utilize a long pulse to achieve large radiated energy, but simultaneously to obtain the range resolution of a short pulse. This is accomplished by using a coded signal instead of a simple CW pulse. At the receiver the returned signal is correlated with a replica of the coded transmit signal. The returned signal would only correlate heavily with the replica for a short time, corresponding to when the echoes are aligned with the replica. This results in a narrow pulse appearing at the output of the correlator, everytime a match occurs. A signal that is commonly used in pulse-compression techniques is the lineal FM signal. An example of such a signal is depicted in Figure 15a. The autocorrelation of such a waveform is shown in Figure 15b. Note that the autocorrelation function has a narrow peak at the origin, with small side lobes elsewhere, i.e. the initially long FM pulse is 'compressed' into a narrow pulse after the autocorrelation process.

It can be shown that the degree of compression is equal to BT where B is the bandwidth of the coded pulse and T is its duration. The effective pulse duration, as far as the range resolution is concerned, will thus be:

$$\text{Effective Pulse Duration} = \frac{T}{BT} = \frac{1}{B} \quad (18)$$

If the 10μ s pulse in the previous example is coded in such a way that its bandwidth becomes 5MHz, the effective pulse duration would be:

$$\frac{1}{5 \cdot 10^6} = 0.2\mu s$$

This corresponds to a range resolution of 30 metres.

VIII.4 System identification using correlation

Another important application of cross-correlation is in the use of random-noise test signals to identify the impulse response of a system. For a system with an unknown impulse response h(t), the output y(t) is related to an input x(t) by

$$y(t) = \int_{-\infty}^{+\infty} h(u)\, x(t-u)dt \quad (19)$$

The cross-correlation between the input x(t) and the output y(t) is defined by

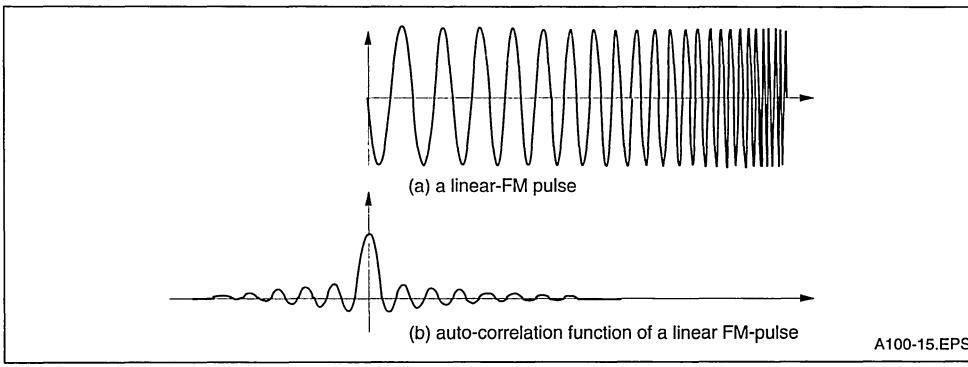
$$\Phi_{xy}(\tau) = \lim_{T \to \infty} \frac{1}{T} \int_0^T x(t)\, y(t+\tau)dt \quad (20)$$

$$= \lim_{T \to \infty} \frac{1}{T} \int_0^T x(t) \int_{-\infty}^{+\infty} h(u)\, x(t+\tau-u)dudt$$

Using simple mathematical manipulation it can be shown that

$$\Phi_{xy}(\tau) = \int_{-\infty}^{+\infty} h(u)\, \Phi_{xx}(\tau - u)\, du \quad (21)$$

Figure 15 : Pulse Compression Function of a Linear FM-pulse



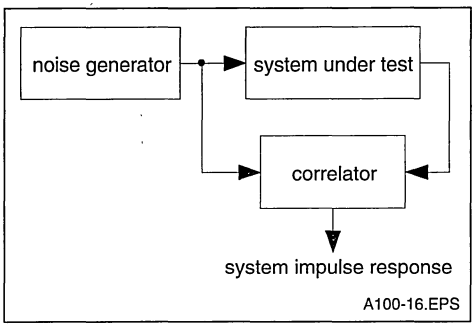
i.e. The cross-correlation between x(t) and y(t) is the convolution of the impulse response h(t) with the auto-correlation of the input signal.

If the input signal consists of broad-band white noise then its auto-correlation function, $\Phi_{xx}(\tau)$, would be an impulse (since a noise signal only correlates with itself at zero delay, $\tau=0$). Referring to equation (21) it therefore follows that for broad-band noise input, the output $\Phi_{xy}(\tau)$ would be a direct measure of $h(\tau)$ since

$$\Phi_{xy}(\tau) = \int_{-\infty}^{+\infty} h(u)\,\delta(\tau - u)\,du = h(\tau) \quad (22)$$

Figure 16 illustrates the technique.

Figure 16 : System Identification using Correlation



VIII.5 The Discrete Fourier Transform (DFT)

DFT has application in many signal processing areas, including speech processing, radar, sonar, image processing and control. This transform has often been performed using Cooley and Tukey radix-2 FFT algorithm (reference 3). This algorithm reduces the number of multiplications compared to direct evaluation of the DFT at the expense of complicating the required indexing.

Other algorithms have also been developed which allow the evaluation of the DFT via correlation (convolution) techniques (references 1, 2, 4, & 5). The IMS A100 device can be used to perform high speed DFT's based on these convolutional algorithms. Using the IMS A100 as a peripheral to a general-purpose microprocessor converts a slow host into a high-performance DFT processor. A separate application note available from INMOS describes how these algorithms can be implemented using the IMS A100 devices.

IX. REFERENCES

1 Burrus C.S., 'Index mappings for multidimensional formulation of the DFT an convolution',
IEEE Trans. on ASSP, Vol.25, June 1977, pp 239-242.

2 Burrus C.S., Parks T.W., 'DFT/FFT and convolutional algorithms-theory and implementation',
Wiley-interscience Publication, New York 1985.

3 Cooley J.W., Tukey J.W. 'An algorithm for the machine calculation of complex Fourier series',
Math. Comput., Vol.19, pp 297-301, April 1965.

4 Rader C.M., 'Discrete Fourier transforms when the number of data samples is prime',
Proc. IEEE, Vol.56, pp 1107-1109, June 1968.

5 Rabiner L.R., et al, 'The chirp z-transform algorithm and its application',
The Bell System Technical Journal, May-June 1969, pp 1249-1292.

COMPLEX (I&Q) PROCESSING WITH THE IMSA100

1. INTRODUCTION

Complex processing, involving in-phase and quadrature signal components, is necessary in many signal processing applications. This type of processing is needed in cases where the phase of the signal has significant impact on the processing outcome. For example consider a simple demodulator as shown in Figure 1. The incoming tone, (A cos ω t), is demodulated by mixing it with a local oscillator having the same frequency. The output of the mixer is low-pass filtered to yield the final result. If there is a phase difference Φ between the incoming tone and the local oscillator signal, the output would be proportional to cos Φ. This indicates that the output of our simple demodulator is strongly dependent on the relative phases of the incoming and local oscillator signals. For the worst case of $\Phi=\frac{\pi}{2}$, the output would become zero! This means that the relative phase shift of the local oscillator can be quite disasterous.

Let us now perform the same demodulation using complex processing. The input signal can be represented in its complex form consisting of real and imaginary parts i.e.

$$x(t) = Ae^{-j\omega t} = A\,[\cos(\omega t) - j\sin(\omega t)] \quad (1)$$

Mixing the signal with a complex version of our local oscillator signal i.e. $A^{j(\omega t+\Phi)} = \cos(\omega t+\Phi)+j\sin(\omega t+\Phi)$ yields:

$$Ae^{-j\omega t}\, e^{j(\omega t+\Phi)} = Ae^{j}\Phi = A\cos\Phi + jA\sin\Phi \quad (2)$$

Note that this output is complex and contains both phase (Φ) and amptitude (A) information. The amptitude can be extracted by taking the modulus of the output i.e.

$$\text{amplitude} = \sqrt{(A\cos\Phi)^2 + (A\sin\Phi)^2} = |A| \quad (3)$$

The above example illustrates how complex processing can be used to preserve both phase and amptitude information in a simple demodulator.

Similar phase related problems arise in correlation and convolution evaluations where complex processing becomes necessary for preserving the integrity of signals. This application note describes how on-chip facilities of the IMSA100 transversal filter can be used to perform complex correlation, convolution and filtering.

As described in the data sheet, the IMSA100 transversal filter incorporates two sets of coefficient memories (Figure 2), each containing 32 16-bit words. At any instant one set of coefficients is applied to the multiply-accumulate array, whilst the other set can be accessed via the IMSA100 standard memory interface. The function of the two memory banks can be interchanged by performing a write operation to the 'Bank Swap' bit of a control register.

This allows the new set of coefficients to be used in the computation at the beginning of the next cycle. In this operation once the two memory banks are interchanged, the 'Banks Swap' control bit is reset by the device. No more interchanges are performed unless the bank swap control bit is again set by the host.

There is another control bit in the static control register of the IMSA100, that when set continuously interchanges the two memory banks at the beginning of each and every computation cycle. When this mode is set, alternate coefficient memory banks will be used for even and odd computation cycles. This mode is particularly suitable for implementing complex data processing. The following two sections describe how this continuous-swap mode can be employed to perform complex convolutions and correlations using the IMSA100 transversal filters. A separate application note, available from INMOS, deals with the correlation and convolution concepts and their implementation using the IMSA100 device. Readers unfamiliar with the IMSA100 and its implementation of correlation and convolution functions are advised to refer to that application note before reading the following sections.

Figure 1 : Simple Demodulator

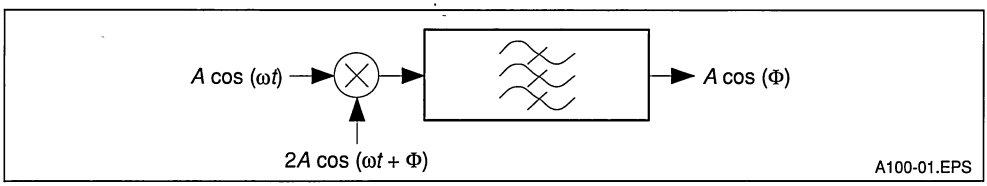
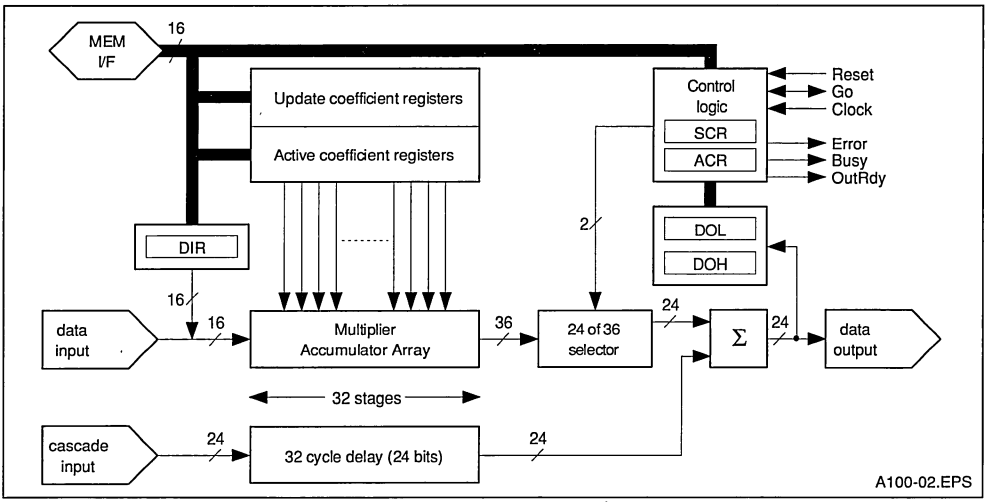


Figure 2 : User's Model of the IMSA100



2. COMPLEX CORRELATION

The complex correlation between two signals r and s is very similar to real correlation (refer to the application note entitled 'Correlation and convolution with the IMSA100') with the difference that one of the two signals has to be complex conjugated first i.e.

$$R_{rs}(m) = \frac{1}{N}\sum_{k=0}^{N-1} r^*(k)s(k+m) \quad (3)$$

or

$$R_{rs}(m) = \frac{1}{N}\sum_{k=0}^{N-1} r(k)s^*(k+m) \quad (4)$$

where * indicated complex conjugate operation and both waveforms r and s can be complex.

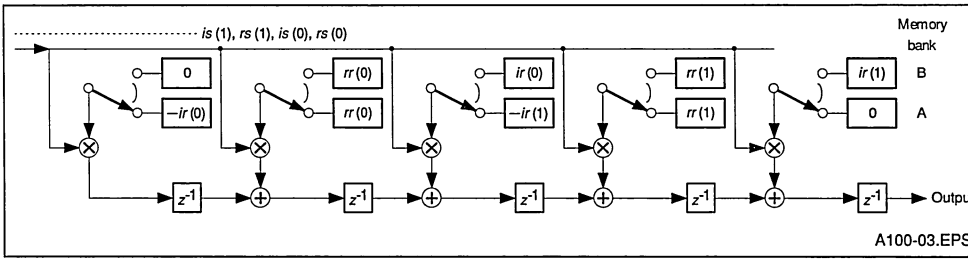
Let us now investigate how the IMSA100 can perform this function. As shown in Figure 2, the computational core of the IMSA100 contains an array of 32 multiply-and accumulators. In order to simplify the explanation of complex processing, let us consider a simple five-stage transversal filter as shown in Figure 3. Once you have understood how such a simple structure can be used for complex correlation, it should be easy to extend the idea to larger correlations sizes involving one or many cascaded IMSA100 devices.

Suppose we want to perform a two-point complex correlation between a reference signal and a sequence of complex input samples. Let us denote the two complex samples of the reference signal with

$$r(0) = rr(0) + j\,ir(0) \text{ and}$$
$$r(1) = rr(1) + j\,ir(1)$$

where rr and ir indicate real parts and imaginary parts of the reference signal respectively.

Figure 3 : Two-point Complex Correlator Based on the IMSA100 Architecture



Assume the input sequence is the following set of complex samples:

$$s(0) = rs(0) + j\,is(0),\ s(1) = rs(1) + j\,is(1)$$
$$, ..., s(n) = rs(n) + j\,is(n), ...$$

where rs(n) and is(n) indicate real and imaginary parts of the nth input sample, s(n), respectively.

The 5-stage transversal filter shown in Figure 3 can be used to correlate these two sequences. The reference signal samples are first complex conjugated i.e.

$$r^*(0) = rr(0) - j\,ir(0) \text{ and}$$
$$r^*(1) = rr(1) - j\,ir(1)$$

These samples of r^* are then allocated to the two coefficient memory banks as shown in Figure 3. It can be seen from this diagram that both coefficient stores contain real and imaginary samples of the reference signal.

Assume that the input sequence is sampled into the correlator, with the real part followed by the imaginery part of each input sample. i.e. the input to the correlator is

$$rs(0), is(0), rs(1), is(1), rs(2), is(2), ...$$

where rs(0) is the first input sample. Also assume that we have selected the continuous-swap mode so as the memory banks A and B are swapped every time a new input is sampled. (On the IMSA100, you can select this mode by writing to a control register). Assuming that the coefficient bank 'A' is selected for the first input sample, B for the second and so on, you should be able to convince yourself that the output sequence for the arrangement in Figure 3 is as shown in Table 1. Note that in this example it is assumed that the correlator is cleared first by writing several zero's to the input.

The last column in table 1 expresses the output sequence in terms of the complex input and complex reference samples. Examination of the output sequence would indicate that alternate samples correspond to real and imaginary parts of the expected correlation function. The arrangement for the two point correlator of Figure 3, can be generalised to N-point complex correlation. Figure 4 illustrates the allocation of a reference signal to the coefficient memories of the IMS A100 for a 15-point complex correlation. The 15 complex samples of the reference signal are represented by:

$$r(n) = rr(n) + j\,ir(n) \text{ for } n = 0 \text{ to } 14$$

where rr(n) and ir(n) are the real and imaginery parts of the nth sample of the reference waveform. Similar to the 2-point complex correlator described earlier, the correct operation is achieved if each input sample is supplied to the chip with its real parts followed by its imaginery parts. The coefficient memories, of course, should be set to the continuous-swap mode.

In general for an N-point correlator realized with the IMS A100 chip, the first sample will always be zero (see table 1). The following N-1 output-sample pairs (real and imaginary parts) correspond to partial results for the following complex correlation coefficients:

$$R_{sr}[-(N-1)], R_{sr}[-(N-2)], ..., R_{sr}(-1)$$

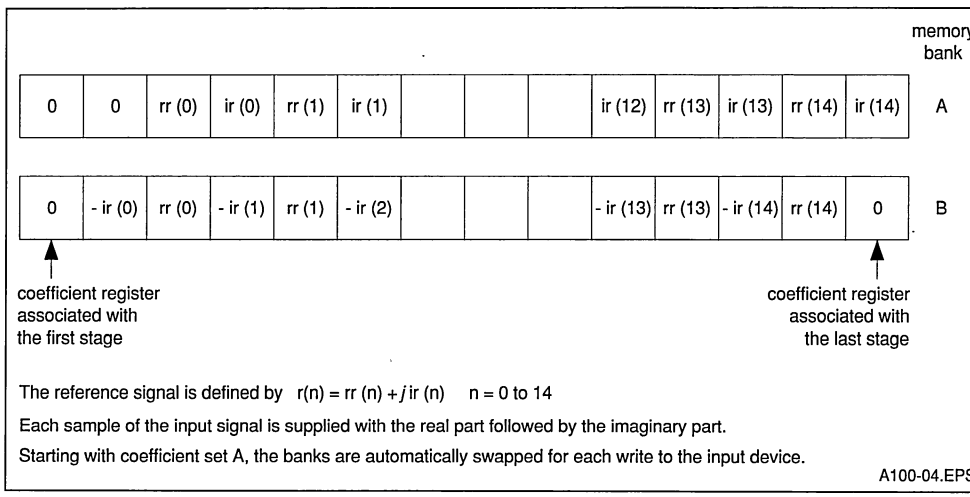
and these will be followed with fully formed correlation coefficients:

$$R_{sr}(0), R_{sr}(1), R_{sr}(2).....$$

Table 1 : Output Sequence for Figure 3

Sample number	Input sample	Output sample value		
1	rs(0)	0	→	0
2	is(0)	rs(0)× rr(1)+is(0)× ir(1)	→	Real part of s(0) x r*(1)
3	rs(1)	-rs(0)× ir(1)+is(0)× rr(1)	→	Imag. part of s(0) x r*(1)
4	is(1)	rs(0)× rr(0)+is(0)× ir(0)+ rs(1) × rr(1)+is(1)× ir(1)	→	Real part of s(0) x r*(0)+s(1) x r(1)
5	rs(2)	-rs(0)× ir(0)+is(0)× rr(0) -rs(1) × ir(1)+is(1)× rr(1)	→	Imag. part of s(0) x r*(0)+s(1) x r*(1)
6	is(2)	rs(1)× rr(0)+is(1)× ir(0)+ rs(2) × rr(1)+is(2)× ir(1)	→	Real part of s(1) x r*(0)+s(2) x r(1)
7	rs(3)	-rs(1)× ir(0)+is(1)× rr(0) -rs(2) × ir(1)+is(2)× rr(1)	→	Imag. part of s(1) x r*(0)+s(2) x r*(1)
8	is(3)	rs(2)× rr(0)+is(2)× ir(0)+ rs(3) × rr(1)+is(3)× ir(1)	→	Real part of s(2) x r*(0)+s(3) x r(1)
9	rs(4)	-rs(2)× ir(0)+is(2)× rr(0) -rs(3) × ir(1)+is(3)× rr(1)	→	Imag. part of s(2) x r*(0)+s(3) x r*(1)
10	is(4)	rs(3)× rr(0)+is(3)× ir(0)+ rs(4) × rr(1)+is(4)× ir(1)	→	Real part of s(3) x r*(0)+s(4) x r(1)
11	rs(5)	-rs(3)× ir(0)+is(3)× rr(0) -rs(4) × ir(1)+is(4)× rr(1)	→	Imag. part of s(3) x r*(0)+s(4) x r*(1)

Figure 4 : Example of Reference Signal Allocation for a 15 Point Complex Correlation using the IMSA100



Complex correlators involving more than 15-points can be implemented either by cascading several IMSA100 devices or alternatively by using mathematical decomposition techniques to convert a long correlation into several short ones which can then be evaluated using a single device. Although the latter approach would require fewer devices, the processing rate would be less than the cascade arrangement.

The IMSA100 can be cascaded without any external components to achieve correlators involving large number of correlation points. As an example, Figure 5 illustrates how a 31-point complex correlator can be made up by cascading two IMSA100 devices. The allocation of a complex 30-point reference signal to the coefficient memories is also shown in Figure 5. The input sequence, having a format described earlier, is supplied to both devices.

3. COMPLEX CONVOLUTION

The convolution process is closely related to that of correlation. In order to convolve two signals, one of the signals is time reversed and the second signal is then correlated (without complex conjugate operation) with this time reversed waveform i.e.

$$C_{rs}(m) = \frac{1}{N}\sum_{k=0}^{N-1} r(k)s(m-k). \quad (5)$$

The process of convolution is what happens in filters where the output corresponds to a convolution of the input signal and the impulse response of the filter. This is equivalent to correlating (without conjugate operation) time-reversed version of the impulse response with the input sequence.

The IMSA100 transversal filter can be used to perform complex convolution between a reference signal

$$r(n) = rr(n) + j\,ir(n)$$
$$\text{for } n = 0 \rightarrow N - 1$$

and an input sequence

$$s(0) = rs(0) + j\,is(0),\ s(1) = rs(1) + j\,is(1),$$
$$\ldots\ldots\ldots,\ s(n) = rs(n) + j\,is(n),\ \ldots\ldots$$

Figure 6 illustrates how the samples of a reference signal should be loaded in the coefficient memories for a 15-point complex convolution. In a similar fashion to the complex correlator implementation, the waveform to be convolved with this reference is applied to the input of the IMSA100 with the real part followed by the imaginary part. The coefficient memory banks should be set to the continuous bank-swap mode as before.

Again several IMSA100 devices can be cascaded to implement longer complex convolvers.

Figure 5 : Cascading two IMSA100 Devices to obtain a 31 Point Complex Correlator

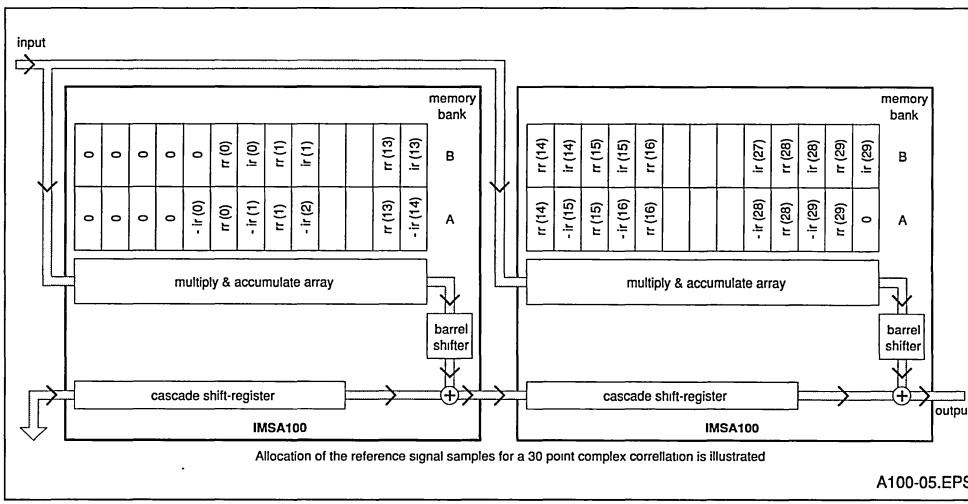
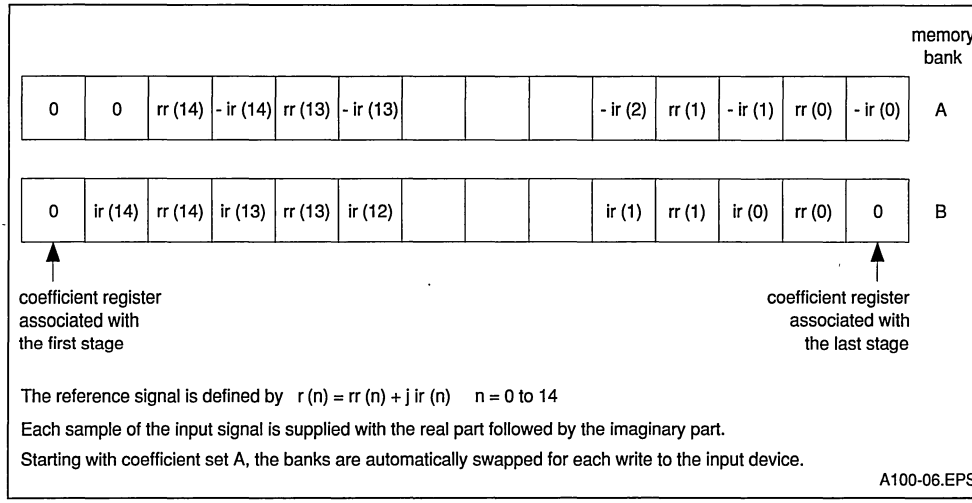


Figure 6 : Example of Reference Signal Allocation for a 15 Point Complex Convolution (filtering) using IMSA100



SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

HARDWARE CONSIDERATION WITH THE IMSA100

SUMMARY

I. INTRODUCTION

The design of modern high speed digital systems is a considerable challenge. If the designer is using unfamiliar new products which themselves are complex VLSI devices then this task can become very difficult. This application note will help those who wish to build systems using the IMSA100 cascadable signal processor.

I.1. Scope of the document

This document should be read in conjunction with the device specification [1]. The device specification is a short, precise, and minimal description of the IMSA100. The following document gives a more detailed description of the function of the device, along with many hints for designing the device into a circuit. Specific hardware designs are also given, which may help the designer further.

I.2. Document summary

In section II a description of the IMSA100 device is given, with a particular emphasis on the operation and timing constraints of the various inputs and outputs.

In section III smaller systems using a few IMSA100 devices are considered.

Section IV describes techniques which allow large and very large systems to be designed without loss of throughput.

Section V describes a method which allows faster data rates to be achieved, by operating several IMSA100 devices in a parallel configuration.

Section VI gives some suggestions for debugging and fault finding hardware after it has been built.

II. THE IMSA100 DEVICE (see Figure 1)

This section gives a functional and parametric description of the device, and should be read in conjunction with the IMSA100 data sheet. The data sheet is a necessary description for design with the IMSA100. The following expands on some of the device mechanics, which are described in the data sheet.

II.1. Pin description and constraints

The description of the various pins of the device is split into a description of power supply pins, asynchronous pins, synchronous pins and control pins

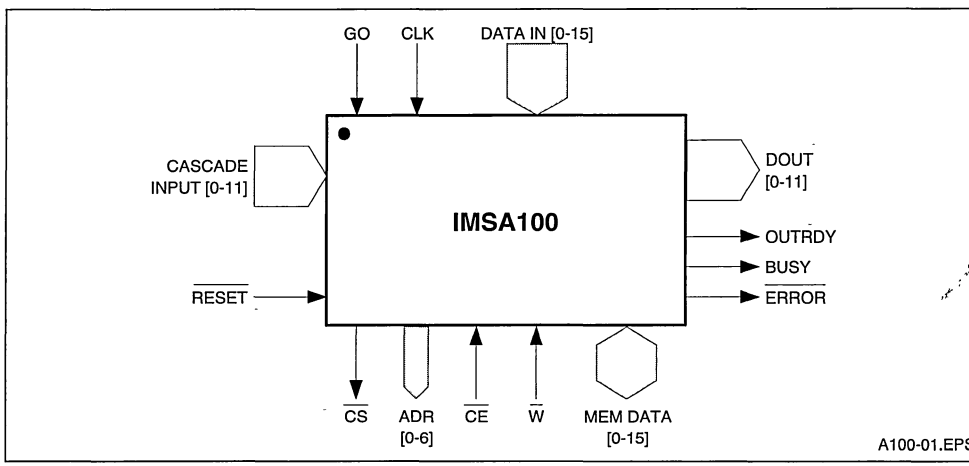
II.1.a. POWER SUPPLY

All power supply pins must be connected to the correct polarity of supply for the device to operate correctly. The supply must be decoupled by a capacitor with a value of 100nF or more which is suitable for high frequencies (e.g. multi-layer ceramic). One or more should be mounted as close as possible to each device and the lead lengths of the capacitors should be minimised. The device is designed to operate with a single supply of between 4.5 and 5.5volts.

II.1.b. SYNCHRONOUS INPUT/OUTPUT

The synchronous pins of the device are CLOCK, GO, OUTRDY, DataIn[0..15], DataOut[0..11] (multiplexed) and CascadeIn[0..11] (multiplexed).

Figure 1 : IMSA100 Device Schematic



The clock

The CLOCK input pin requires a non-standard input signal of >4.0volts for a high level and <0.5volts for a low level. The waveform needs to be monotonic between these two levels. Details of how this is achieved are given in later sections of this application note. In general, a CMOS level clock driver with proper termination will be needed. The CLOCK pin forms a large capacitive load (12pF typ., 15pF max.) which needs to be considered when designing the clock driving system.

The CLOCK is the source of synchronisation between cascaded IMSA100 devices; GO, Data In[0..15] and CascadeIn[0..11] all sample in response to it. The output signals DataOut[0..11] and OUTRDY are also timed from the clock. When the IMSA100 devices are programmed to operate in the normal mode the timing constraints associated with the transfer of data between adjacent cascaded IMSA100 devices is less rigorous than in 4 bit or fast modes. If the devices are being used in fast output or 4 bit modes it is important to keep the timing skew of the clock between devices to a minimum. In practice, it is not a good idea to buffer the clock between devices in these modes. If a master generated GO pulse is being used, a common clock is recommended. The maximum clock frequency for an IMSA100 is normally 20.8MHz, (a 30MHz version device is also available) but the device is fully static and will therefore operate at any frequency below this. It is also possible to start and stop the clock, provided no single phase becomes shorter than the minimum indicated in the data sheet.

GO

The GO pin initiates a compute cycle of the IMSA100 and synchronises the devices in a multiple IMSA100 system. The GO pin is sampled on every rising edge of CLOCK when the IMSA100 is idle, and no computation cycle is in progress [1]. When a '1' is sampled, a computation cycle is started, and the DataIn pins or data input register DIR is sampled on the next rising edge of CLOCK. The GO pin will not be sampled again until it is possible to commence another cycle. It is therefore possible to leave the GO pin at a '1' level following the initial clock edge, if the maximum data throughput is necessary. The number of clock cycles between successive GO samples and the result appearing at the output, are dependant on the coefficient word length setting [1]. CIN is also sampled following a GO signal.

For the GO pin to be an input, the IMSA100 is set to be a slave by setting SCR[0] to a '0'. However, the IMSA100 can be programmed to provide a GO signal for itself and for other IMSA100s by setting SCR[0] to a '1'. This causes the device to send a signal from it's GO pin in response to a value being written to it's DIR register. The falling edge of this signal indicates when new data can be safely written to the IMSA100s in the system. This feature is particularly useful in small systems where a microproccessor is being used to provide data. It should be noted that the master IMSA100 is only designed to drive itself plus another 3 devices (maximum load < 20pF) when operating at the maximum clock rate. However, at slower clock rates more devices can be added in line with the following table.

Max clock Frequency	Max no of slaves	Length of filter
20MHz	3	128
17.5MHz	4	160
16MHz	5	192
15MHz	6	224
10MHz	10	352
5MHz	30	992

It is possible to buffer the GO signal from the master IMSA100 providing the buffer is fast enough. The propagation delay for such a buffer with a 5pF input capacitance as a function of the IMSA100 clock period is given below.

$$Tp_{buffer} < Tclk - 46ns$$

An alternative method of buffering the GO signal is discussed in the section of this note dealing with large systems.

Data input bus

This 16 bit wide DataIn[0..15] provides high speed data to the IMSA100s when SCR[1] is programmed to '0'. Usually, DataIn[0..15] will be common to all the IMSA100s in a given cascade. The data is sampled on the rising edge of the clock following the acceptance of a 'GO sample' by the devices. If this bus is being used to provide data, the IMSA100 must be in slave mode and cannot be used as a master. Each pin represents a capacitive load of about 5pF.

Data output bus

The 24 bit result from the IMSA100 is multiplexed through DataOut[0..11] as two 12-bit words, the least significant word being first. The most significant word follows and remains on the pins until the next least significant word is available. The timings of the signals are dependant on the coefficient word

length and the normal/fast setting as defined in the SCR. In the 4-bit and fast modes the least significant word is only available on one rising edge of CLOCK, with the most significant word being sent immediatly following the same edge. In the 4-bit mode running at full speed, every rising edge is used to both latch the output data into the CIN pins, and to cause the output data to change to the new value. The advantage of the fast output mode is that the complete 24-bit output word is made available at the earliest possible time, whereas the normal mode delays the most significant word slightly. This eases the timing constraints of the circuitry sampling the output data. All devices in a given cascade must be set to the same coefficient word length and fast/normal option. The output drivers used on the data output pins are designed to drive small loads (e.g. 2 TTL inputs or about 15pF) with a 20MHz CLOCK in the fast or 4-bit modes. Even in the normal mode the load should not exceed 30pF on these pins.

Output ready signal

The output ready pin OUTRDY is provided to indicate when the two 12-bit output words from the data output pins are valid. It can be used to demultiplex the output into registers, and also indicates when the data has been stored in the data output registers DOL and DOH. The falling edge of the OUTRDY signal indicates that the least significant word on the output is valid, whilst the rising edge indicates that the most significant word is valid and that the DOL/DOH registers contain the new output data value. As in the case of [DataOut[0..11] the timings of this signal are dependant upon the coefficient word length and the fast/normal mode setting. Again, the timing constraints are eased when the device is operating in the normal mode on 8-bit, 12-bit and 16-bit coefficient sizes. In the fast or 4-bit modes the OUTRDY signal is triggered by the falling edge of CLOCK following the rising edge of CLOCK which changes the output data. The OUTRDY pin should have a similar loading to the data output pins for optimum timing, and this should not exceed the limits set for the data output pins.

The OUTRDY signal can be used to supply a clock for a D/A converter which, if it uses less than 12 of the available 24 bits, will only require one of the two 12-bit words, thereby avoiding the need for demultiplexing logic. When demultiplexing is required, it can be achieved using two sets of edge triggered latches (e.g. 74ACT374) which are clocked by OUTRDY and it's inverse (Figure 1). It is suggested that any external logic associated with the DataOut[0..11] and OUTRDY pins be of a fast TTL compatible CMOS logic type (i.e. FACT) in order to minimise loadings.

Cascade input port

The cascade input port allows multiple IMSA100 devices to be cascaded together in a chain. Like DataOut[0..11] , CascadeIn[0..11] is 12 bits wide and two words are used to form a 24-bit word with the least significant word being sampled first. The cascade input timings are given in the device specification, but it should be noted that the OUTRDY signal does **not** normally coincide with the sampling of CascadeIn[0..11] . The cascade input of the first device should be grounded unless data is to be supplied to it.

II.1.c. MEMORY INTERFACE
ASYNCHRONOUS INPUT/OUTPUT

The memory interface is the asynchronous part of the system. It is designed, as far as is practical, to appear as a memory mapped peripheral. To achieve this there are chip select, chip enable, read/write and address and data bus signals, which will now be described.

Chip select pin

The chip select pin $\overline{CS}$ has to be pulled low (active) at the appropriate time for the memory interface to be enabled. This pin is usually connected to part of an address decode system.

Chip enable pin

The chip enable pin $\overline{CE}$ is pulled low (active) to enable the memory interface, after the address, write enable $\overline{W}$ and chip select $\overline{CS}$ signals have been set up.

Read not write

The read not write signal $\overline{W}$ defines whether a given cycle is reading from or writing to the IMSA100 memory. This signal should not be changed whilst $\overline{CE}$ is low.

Memory address bus

This 7 bit wide port ADR[0-6] is used to address the IMSA100 memory. A memory map is given in the IMSA100 specification showing locations of the two coefficient banks and control registers. The TCR register, located at decimal address 68, will default to all zeros on power up or in response to a $\overline{RESET}$ signal. However if it is disturbed, by for example a system memory test, it should be written back to all zeros. Failure to do so may result in unpredictable results.

Memory data bus

The 16 bit wide memory data port DATA[0-15] handles both input and output data to and from the IMSA100 and is used to program the two banks of coefficient registers and the control registers. When writing to a coefficient register, the memory interface is transparent while $\overline{CE}$ is low. Writing to the active coefficients whilst the IMSA100 is running a computation cycle may cause an incorrect transient coefficient to be used. Using the update registers followed by a bank swap avoids this problem.

When the $\overline{CE}$ or $\overline{CS}$ are high the data pins are tri-state. The output stages associated with the data pins are current limited and may be loaded by more than the 30pF specified for the timings given in the specification provided the $\overline{CE}$ pulse length is increased. The table below gives an indication of the length $\overline{CE}$ pulse required for a series of loads.

Capacitive load on data bus	$\overline{CE}$ pulse length
300 pF	50 ns
100 pF	80 ns
300 pF	170 ns
1000 pF	500 ns

Each Data pin represents a maximum load of about 7pF when tri-stated.

II.1.d. SYSTEM CONTROL

The IMSA100 is controlled by 3 signals, $\overline{RESET}$, $\overline{ERROR}$ and $\overline{BUSY}$ which will now be described.

Resetting the device

To reset the IMSA100 control logic pull the $\overline{RESET}$ pin low for at least 200ns followed by two cycles of the input clock. The reset function on the IMSA100 only resets the control registers to their default values. It does not change the values of the coefficients, clear the data path or reset the error flags if errors are still present. There is a power on reset signal ORed with the $\overline{RESET}$ pin circuitry which requires an adequate voltage on both the power supply and the internal clocks before it allows the internal reset signal to fall. For this reason the CLOCK pin must be exercised at or following power up before the control registers are programmed. A resistor (e.g. 33kW) from the $\overline{RESET}$ pin to V_{CC} together with a capacitor (e.g. 10µF) to GND is usually sufficient to provide an adequate signal. The pin may be connected directly to V_{CC} provided you are sure that your system power supply is monotonic on power up, as the power on reset circuit will only operate once.

Error control

If the $\overline{ERROR}$ pin is asserted it indicates that there has been a numerical overflow in either the final adder or field selector. Bits [1-2] in the Active Control Register ACR indicate which error type and the $\overline{ERROR}$ pin is reset by writing a '0' to these registers. Before continuing, these error bits must be armed by writing a '1' to bits[1-2] of the ACR. The $\overline{ERROR}$ pin is only able to sink current to GND and therefore requires a pull up resistor to Vdd. Many devices can be wire ORed together to indicate an error in any one of many IMSA100 devices within a given system. The presence of an error does not affect the operation of the IMSA100 (although the results may be nonsense) and it is possible to continue to use the device without resetting the condition. Although the ACR register is reset on power up, the $\overline{ERROR}$ pin is usually set again by random numbers within the device. In order to clear the $\overline{ERROR}$ pin it is necessary to flush the system before clearing and re-arming the ACR registers. Flushing involves writing zeros to the data input and cascade input over 32 successive cycles.

Device busy

The $\overline{BUSY}$ pin indicates when the active and update coefficient registers are being or are about to be swapped. When this pin is high the coefficient registers should not be accessed. There is no guaranteed minimum duration of a $\overline{BUSY}$ signal since a bank swap request may be dealt with immediately. This pin operates only in conjunction with individual bank swap requests made via ACR[0] and not when the continuous bank swap mode is selected by SCR[2].

II.2. Initialisation of IMSA100s

There are many ways of initialising one or more IMSA100 devices but if in doubt the following procedure is recommended. First, for a system with all devices used as slaves do the following operations.

1. Apply power and start CLOCK
2. Take the $\overline{RESET}$ pin high
3. Write all coefficients to '0'
4. Set the CascadeIn[0..11] of the first devices in any cascades to '0'
5. Set GO to '1' or provide plenty of GO pulses
6. Allow the system to run like this for long enough to clear out any stored junk numbers. This period will depend on the length of your filter and the frequency of your GO pulses.

7 Apply a $\overline{\text{RESET}}$ signal or write '0s' and then '1s'to the ACR[1-2] --- any error signal should now disappear.
8 Set up your own SCR

For a system with a master and slaves do the following.

1 Apply power and start CLOCK
2 Take the $\overline{\text{RESET}}$ pin high
3 Set up your own SCR values
4 Write all coefficients to '0'
5 Set the CIN of the first devices in any cascades to '0'
6 Write to the data input register DIR on the master IMSA100 to create plenty of GO pulses
7 Allow the system to run like this for long enough to clear out any stored junk numbers. This period will depend on the length of your filter and the frequency of your GO pulses.
8 Write '0s' and then '1s' to ACR[1-2] --- any error signal should now disappear.
9 Set up your own coefficient and data values.

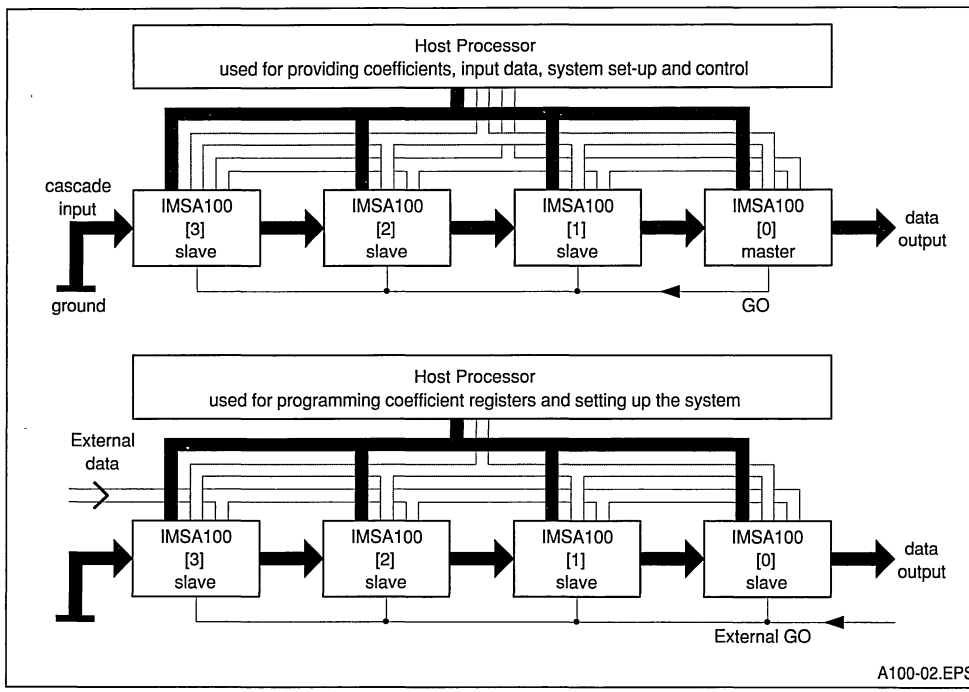
II.3. An extra selector setting using TCR

The test control register TCR is designed to help INMOS fully test the IMSA100. However, one of it's functions may be of interest if the output word selection field of [7-30] gives insufficient resolution. This is most likely to occur when smaller coefficient word lengths are in use. Writing a '1' to TCR[2] will overide the values programmed in SCR[4-5] to give a field selection of [-1-23] where field bit [-1] will always be '0'. The other TCR bits must always be set to '0' by the user.

III. SMALLER IMSA100 SYSTEMS

The techniques described in this section apply to systems employing perhaps four IMSA100 devices in a single cascade together with a small support system including a single microprocessor. Two typical systems are shown in Figure 2.

Figure 2 : Two Simple Small Systems



III.1. Board Layout Constraints

During normal operation the IMSA100 dissipates a fairly low average power (0.5W at 20MHz approx). However, due to the high degree of parallelism within the device, it requires well decoupled, low inductance connections to it's power pins. A multi-layer board with a V_{CC} and GND plane is recommended with at least one multi-layer ceramic decoupling capacitor of 100nF or more mounted as close as possible to each IMSA100. The IMSA100 devices forming a cascade should be located next to each other with the CascadeIn[0..11] pins of the next device near the DataOut[0..11] pins of the first. Any circuitry using the DataOut[0..11] pins should be located as near to the IMSA100 as possible to avoid excessive loading. The track carrying CLOCK should take a direct route from one IMSA100 to the next in order to avoid excessive skew.

III.2. Memory Interface

The last IMSA100 in a cascade chain should occupy the lowest address space and the first in the cascade the highest. This is to maintain compatability with the addressing of the coefficient registers where coeff[0] resides in the lowest location within the bank.

In many applications it will not be necessary to buffer the pins associated with the IMSA100 memory interface. It is, however, necessary to confirm that this is, in fact, the case. The timings of the proposed memory interface together with the bus loadings should be checked with the IMSA100 device specification and the additional information given in Section 2 of this note. If the memory interface uses high speed buffers, some termination may be required to limit transients outside the power supply rails. In such cases 100W resistors in series with the offending buffer(s) are recommended.

III.3. Clocking

In general, the smaller the system, the easier the clocking will be. The IMSA100 CLOCK is not TTL compatable and will have to be generated by a device or devices capable of driving signals to within about 0.5volts from each power rail. The constraint that the CLOCK signal at the IMSA100 should be monotonic in between the high and low limits will almost certainly mean that termination will be required. The easist way of generating such a CLOCK for a small system is to use a TTL compatible CMOS device such as a 74ACT244 in the FACT family of devices. A chain of IMSA100 devices connected by a 10 thou 100W impedance track and driven from one end will need a terminating resistor of about 39W at the other (Figure 4). The exact values of terminating component will depend on many factors and the values given here are for guidance only and some experimentation may be needed. In systems using a slow CLOCK rate a slower CLOCK edge may ease or even remove the termination constraints.

III.4. Data input

The input data can be provided either through the memory interface to the DIR register or through the 16 DataIn[0..15] pins. The main constraint on the DataIn[0..15] pins is that the data should meet the set up and hold times given in the device specification for the relevant CLOCK edge. In the majority of cases DataIn[0..15] will be common to all IMSA100 devices. Termination on the drivers of this bus may be necessary under certain circumstances.

III.5. Data output and output ready

The output data can be obtained from the DOL/DOH registers via the memory interface or from the 12 DataOut[0..11] pins. When the DataOut[0..11] pins are used the two 12 bit words will have to be separated in some way. In systems where less than 12 bits of the answer are required (e.g. to drive an 8 bit D-A converter) it may well be possible to discard one of the two words by choosing appropriate coefficient values and/or selector settings. The OUTRDY signal can be used as a basis for a CLOCK for a D-A converter or other circuitry but it will need buffering if the load on it becomes excessive. If the full 24 bits are required the OUTRDY signal can be used to CLOCK edge sensitive latches [1].

III.6. Master generated GO

The master generated GO feature of the IMSA100 was designed principly for small systems where the input data is supplied through the memory interface. On a master device, the GO pin has a dual function; first to provide a GO signal for all the IMSA100 devices, and second to indicate to the system when it is appropriate to write a new value to DIR and hence start another cycle. It is difficult to obtain a high throughput, compared with using DataIn[0..15] , if the output is being read from the DOL/DOH registers, particularly in the 4-bit and 8-bit modes. Care is needed to avoid writing a new value into the DIR before the old value has been

used, (the correct time is indicated by the falling edge of GO) or reading the DOL/DOH registers while they are being updated (the correct time is indicated by the rising edge of OUTRDY). In a multiple IMSA100 system using a master it is necessary to update the slave DIR registers before, or at the same time as, the master. It does not matter which device is the master but there must only be one for a given cascade. It is possible to update the DIR registers of all the IMSA100 devices by addressing all their DIR registers simultaneously by pulling all the $\overline{CS}$ pins low during the write to the master's DIR. Alternatively, the input data can be provided to all the slaves via the common DataIn[0..15] which, in order to be safe, will have to remain valid until the rising edge of the CLOCK following the falling edge of the master generated $\overline{GO}$ signal. In this case the SCR registers in the slaves must be programmed to accept input data from DataIn[0..15] and not the DIR register. It is not possible for the master IMSA100 to take it's data from DataIn[0..15].

III.7. External GO

For high speed systems and for all systems where the input data is only provided via the DataIn[0..15] port a GO signal must be provided by the support system. In many systems where the maximum throughput is required the GO signal may be taken high but it is important to keep track of when DataIn[0..15] is sampled to avoid changing the input data at this time. The GO signal may be pulsed every N CLOCK cycles at or less than the maximum data rate but any attempt to pulse GO at a higher rate will result in a drop in speed due to some of the pulses being ignored. It is important that the GO signal changes outside the set up and hold times given in the device specification to avoid the risk of different IMSA100 devices in the cascade falling out of synchronisation. If IMSA100 devices in a cascade do get out of synchronisation with each other for any reason, they will immediately resynchronise on a new correctly timed GO signal.

IV. LARGE IMS A100 SYSTEMS

This section deals with design techniques suitable for overcomming the problems raised when designing systems employing many IMSA100 devices. There is a limit to the number of IMSA100 devices that can easily be put in a single cascade without break and that limit will depend on many factors but especially board size and speed. Many of the problems are the same as those already dealt with in the previous section but more severe. Whilst with a small system it is fair to assume that every IMSA100 is on a single board, this may well not be the case with large systems. However, with care it is possible to build very large systems of IMSA100 devices with a phenomenal performance.

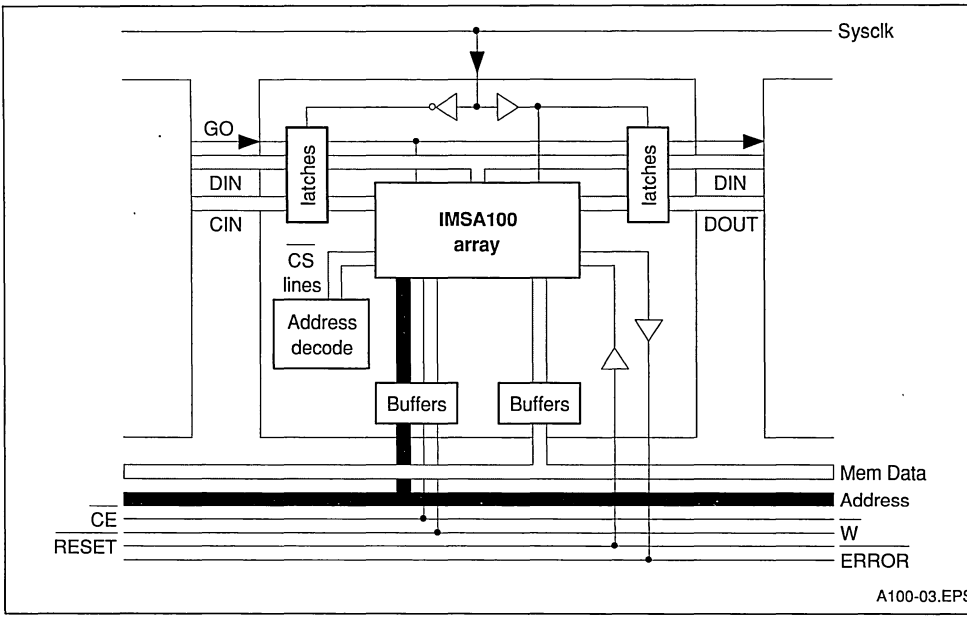
IV.1. How many IMSA100 devices per board ?

In theory it is possible to put as many IMSA100 devices in a continuous chain as necessary without limit providing all the signals to and from the devices meet the specification. In practice boards have a finite size, bus capacitances build up to unreasonable values, and so on. It therefore becomes necessary to partition the problem. In practice it is possible to put 32 IMSA100 devices on a double extended Eurocard, using a 4 layer printed circuit board together with enough additional logic to allow these boards themselves to be cascaded. Such a board could be regarded as a 1024 stage subsystem. This same technique can be applied to smaller numbers of IMSA100 devices on smaller boards, although address decoding will be easier if 32, 16, 8 or 4 devices are grouped together. Whilst the data throughput of the cascade can be maintained, the speed of the memory interface will be a funtion of the loading of the data lines. For many applications this will not matter but in applications, such as fast adaptive filtering, the rate at which the coefficients can be updated may be important. It is therefore necessary to identify which aspects of performance are important, as they will have a significant effect on the way that the system is implemented.

IV.2. Cascading boards

This section describes one way of maintaining the maximum throughput of the IMSA100 devices in a multiple board system by the use of pipelining. The general technique is to contain the timing problems to each board separately and to make inter-board communication as easy as possible. Each board has a series of edge triggered latches (e.g. 74374 devices) which latch all syncronous inputs and outputs including DataIn[0..15] and GO. In principle, all inputs are latched by a PH1 clock which is inverted to provide a PH2 clock to latch the outputs and to provide the clock for IMS A100s. The signal is transmitted between boards between the rising edge of PH2 and the rising edge of PH1 with the latches acting as drivers (Figure 3).

Figure 3 : Placement of IMSA100 into Large System



IV.3. Board Design

The design of large boards in a cascade requires some care. This section considers the specific example of a cascadable board with 32 IMSA100 devices and support circuitry designed to run with a 20MHz clock with 4, 8, 12 or 16-bit coeficient word lengths. Each of these boards represents a 1024 stage filter and all inputs and outputs are latched or buffered.

IV.3.a. BOARD DESCRIPTION

To minimise the length of the connections between DataOut[0..11] and CascadeIn[0..11] of adjacent devices, it is best to arrange the IMSA100 devices in a pattern like a snakes and ladder board (Figure 4). This has the additional advantage that common signals like GO, CLOCK, and the various buses may be shared between two rows of devices.

To maximise the density of devices within the board area, half of DataIn[0..15] and half of the memory data bus pass under each row of devices. The address bus and other signals pass between the first and second rows and third and fourth rows whilst CLOCK and GO pass between the second and third rows and the fouth and fifth rows respectively (Figure 4).

The block of IMSA100 devices are mounted away from the board edge connectors. The Cascade input latches drive the top of the IMSA100 block, whilst the output data is produced at the bottom where it is latched. The input data is pipelined to drive the next board as well as providing data for the IMSA100 DataIn[0..15] . The GO signal is pipelined in a similar way to provide a delayed GO signal in step with the delayed input and output data. The system clock is used to provide PH1 and PH2 signals from two CMOS inverting buffer devices located centrally on the connector side of the board. The clock is distributed to keep the timing skew on the clock to adjacent devices to a minimum. The clock tracks driving the IMSA100 devices are terminated by 39W for the top track and 27W for the rest which are driving two rows of devices. The termination consist of a resistor in series with a 100nF capacitor to remove any DC path to GND. The tracks carrying the clock between IMSA100 devices were 10 thou wide, but the tracks from the clock driver to the beginning of the block of IMSA100 devices were of a width needed to match the terminating impedance. The GO signal is treated in a similar manner to the CLOCK , with the option of connecting termination components at the

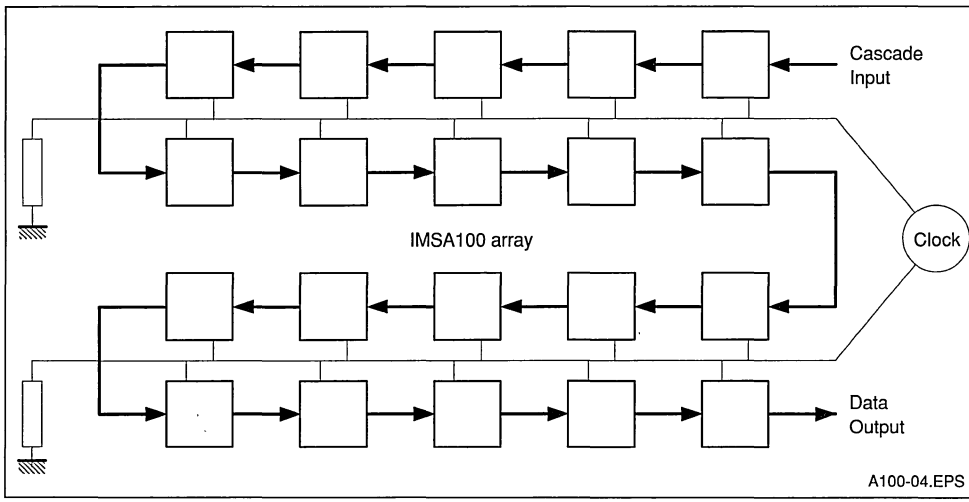
end of each track. The only signals which are not driving every device are the DataOut[0..11] to CascadeIn[0..11] links, and the CS connections. Each one of these is connected separately to the address decoder.

The memory interface buses, the $\overline{\text{ERROR}}$ and $\overline{\text{RESET}}$ functions are not latched but buffered in some way. The ADR[0-6], $\overline{\text{CE}}$ and W signals are all buffered from the memory interface. The data bus passes through a bi-dirctional buffer (74F245 in this case) with it's direction defined by the RnotW signal and with it's tri-state control connected to the board address decoder. Since there are 32 IMSA100 devices together with about 2 feet of pcb track attached to each memory data pin, the CE signal needs to have about 150ns duration.

The $\overline{\text{RESET}}$ pins of the IMSA100 devices are connected together and connected to some open collector logic plus a resistor to V_{CC} and a capacitor to GND. This arrangement allows a $\overline{\text{RESET}}$ signal to be applied from the system but allows the board to reset itself if no such signal is applied. An LED indicates when the $\overline{\text{RESET}}$ signal is high.

The $\overline{\text{ERROR}}$ pins are also connected together with a 1K\Omega resistor pulling up to V_{CC}. This signal is buffered with an open collector gate to allow the boards to be wire ORed if desired. A second LED indicates if an error has occured in any IMSA100 on the board.

Figure 4 : Clock Distribution for Large System



IV.3.b. MEMORY MAPPING

The exact memory map required will vary from system to system but there are one or two pitfalls which should be avoided. The coefficient registers in the IMSA100 are addressed in such a way that the last coefficient is located at address[0] and the first in location [31]. In order to be consistant with this, the LAST IMSA100 on the LAST board should occupy the LOWEST memory location. Failure to implement this will make the block moving of stored coefficient values to the IMSA100 devices less straightforward than it could have been. If the coefficient registers are to be in a continuous memory space, it is necessary to organize the memory map as follows:

```
System Address bits:     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
IMS A100 address bits:   0  1  2  3  4  5                       5  6
Chip Select addresses:                  0  1  2  3  4
Board Select addresses:                                 0  1  2  3
```

This map assumes a 16 bit system address bus and 32 IMSA100 devices on each board. Remember that the board address space should be large enough to cover both the number of IMSA100 boards required plus any other areas of circuitry requiring address space (e.g. memory etc). 5!UW~46IKE

V. HIGHER DATA RATES USING MULTIPLE IMSA100 DEVICES

For some applications, data rates in excess of 10 M samples/sec must be used for real time processing. Since the fundamental maximum data rate of the IMSA100 is 10 M samples/sec, this may appear to be a limiting factor. The following section describes a general method for interleaving multiple IMSA100 devices to achieve effective data rates of 20 M samples/sec and above, with little or no loss of functionality.

V.1. Principle of operation

Figure 5 shows four IMSA100 devices connected so as to provide the equivalent functionality of a 64 stage, 20 M sample/sec IMSA100. The data rate to each device is reduced by introducing a data demultiplexer, which splits the data stream into two parallel streams. This enables a reduction of input data rate to 10 M samples / sec, the maximum possible for a standard IMSA100.

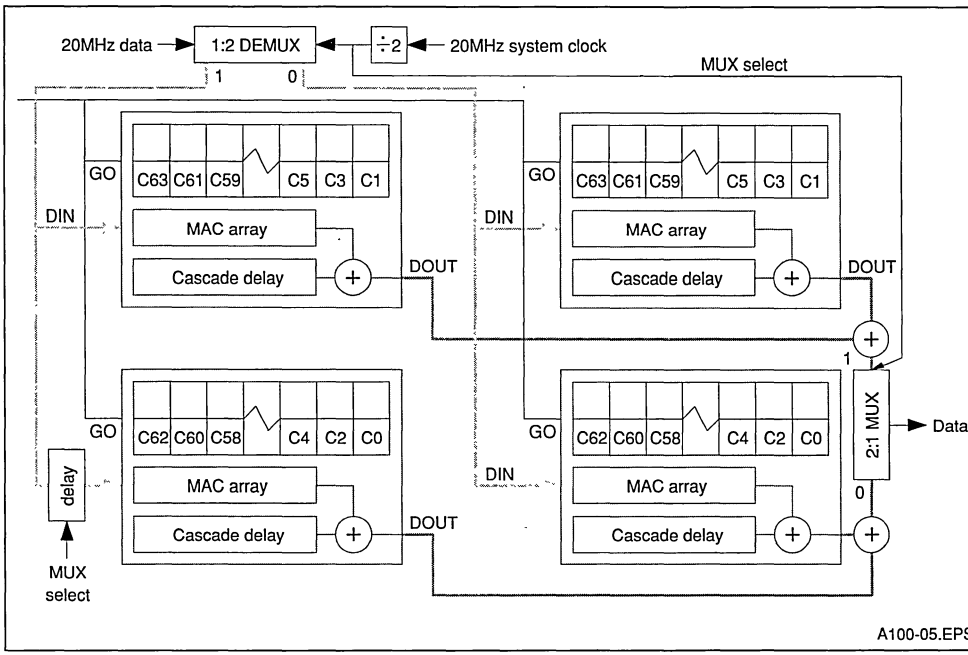
The segmentation of the problem is achieved because of the transversal filter architecture of the IMSA100. For any transversal filter structure, the summation performed at any given time is as follows.

$$C_0x_0 + C_1x_{-1} + C_2x_{-2} + C_3x_{-3} + C_4x_{-4} + ...$$

where x_{-2}, x_{-1},x_0,... represent successive data samples in time, and C_0,C_1,C_2, ... represent the coefficients. Equation 1 can be rewritten as follows, with the two halves of the equation performed by two separate devices. This equation may be further generalised into N segments, executed on N^2 devices.

$$(C_0x_0 + C_2x_{-2} + C_4x_{-4} + ...)+$$
$$(C_1x_{-1} + C_3x_{-3} + C_5x_{-5} + ...)$$

Figure 5 : 20MHz System using External Adders



Figures 5 and 6 illustrate systems exploiting equation 2 to perform 20 M sample / sec filtering. The principle is that the upper pair of devices perform the evaluation for one time period, and the lower devices perform the evaluation for the next, which gives an interléaved computation. In these figures the abbreviations UL refers to the upper left IMSA100, LL the lower left, UR the upper right and LR the lower right.The following points should be noted when observing these figures.

- The positions of the coefficients are in reverse order
- One delay stage is required at the input to the LL device.
- Both evaluations are being performed at exactly the same time, so that the major cycles of all devices commence on the same clock edge. This is set to coincide with the time that a data sample arrives at the UL and LL devices.

V.2. Mechanics of Operation

To see exactly how the circuit works, consider the following sequence arriving at the data input to the demultiplexer.

$$x_0, x_1, x_2, x_3, \ldots$$

Assume that x_0 is sent down bus 0, x_1 is sent down bus 1, x_2 down bus 0, and so on. Consider the major cycle that commences for all devices when x_3 arrives at the UL device. At that time, x_2 will be at UR, x_1 at LL, and x_2 at LR. For this cycle, the final output from UL and UR will be as follows.

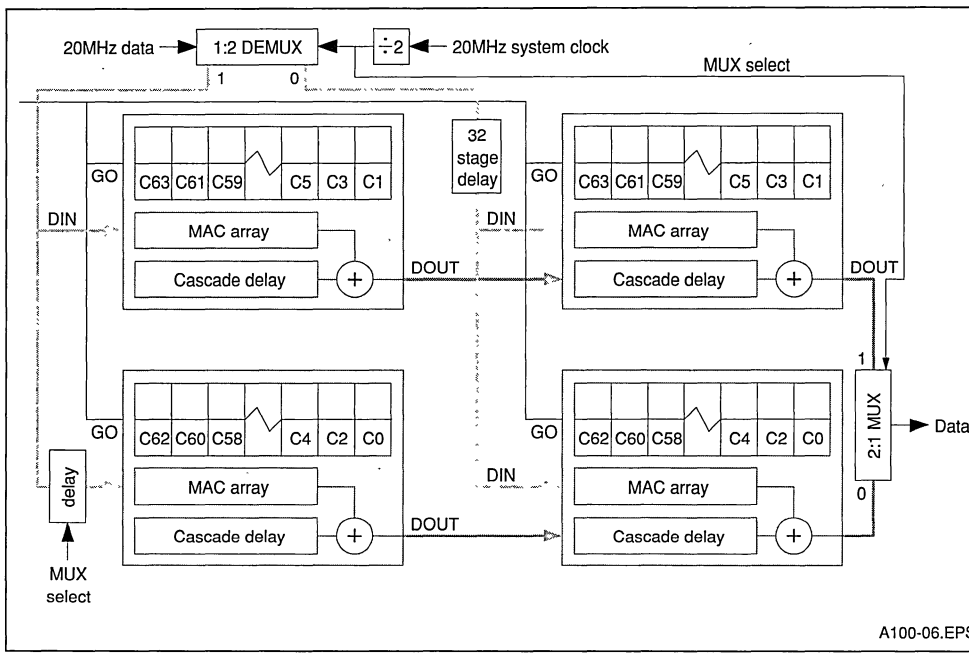
$$(C_0x_3 + C_2x_1 + C_4x_{-1} + \ldots + C_{62}x_{-59})_{UL} + (C_1x_2 + C_3x_0 + C_5x_{-2} + \ldots + C_{63}x_{-60})_{UR}$$

whilst LL and LR will produce the following.

$$(C_1x_1 + C_3x_{-1} + C_5x_{-3} + \ldots + C_{63}x_{-61})_{LL} + (C_0x_2 + C_2x_0 + C4x_{-2} + \ldots + C_{62}x_{-60})_{LR}$$

Thus, the output of the lower pair must be taken first through the output multiplexer, followed by the upper pair. The single delay at the input to LL is necessary for the organisation of coefficients. Because C_0x_n must be the last calculation, data must be presented to the device with coefficient C_0 after the device with coefficient C_1.

Figure 6 : 20MHz System using Cascade Adders



V.3. Using the cascade adders

For most applications, it is more convenient to use the cascade adders rather than external devices (Figure 6). This is because the multiplexed output of the IMSA100 causes complication. A reduction in the number of devices used, and a simplification of the design can be achieved, by using the cascade inputs of the IMSA100.

The cascade adders are used by first inserting a 32 sample delay into the path of the data for the UR and LR devices, and second, by connecting the data output of UL and LL to the cascade inputs of UR and LR respectively. This avoids the use of two accumulator devices, and simplifies circuit board layout considerably. Of the two designs this is the more elegant, and is recommended for use in practice.

V.4. Extensions to this technique

Once the above technique functions correctly, many extensions are possible. Some of these extensions make the design even simpler.

- **Higher speed.** By using a 3×3 or 4×4 configuration, data rates of up to 30 M samples/sec and 40 M Samples/sec respectively can be achieved. The only limitation is the speed of the demultiplexing and multiplexing logic. The minimum number of stages using this method also increases proportionally. Thus for 2×2 devices, a minimum of a 64 stage system is produced, which can only be incremented in 64 stage modules. Likewise for 3×3, the minimum number is 96 stages, and for 4×4 the minimum is 128.
- **Cascading.** Since two cascaded IMSA100 devices appear functionally equivalent to one 64-stage IMSA100, each of the four devices shown can be replaced with N IMSA100 devices to form longer filters. The 32 stage delay would, for example, become 64 stages with two cascaded devices per location.
- **Complex Processing.** The configuration described permits complex processing, using bank swap as described in [5]. However, the two multiplexed data streams presented in the illustrated configurations will be the real and imaginary data streams, and the results likewise. Thus, by providing the complex input data correctly skewed in time, the multiplexers are eliminated. This results in a considerable simplification of the design.
- **Removing** UL delay. The single stage delay can be removed if less than 64 stages are required. This is done by having a zero coefficient in the coefficient closest to the back end (leftmost on the illustration) of the LL device.
- **Removing 32 stage delay.** The 32 stage delay can be eliminated, by zero filling the leftmost coefficients of UR and LR devices. This, although simplifying the circuit, may be costly, as the IMSA100 stages are being used as delay elements. The merits of this depend on the relative cost of an IMSA100 as compared with the cost of a 32 stage delay element.

VI. CHECKING AND DEBUGGING

This section gives some hints on how to check and debug the IMSA100 part of a system. The IMSA100 has a number of features which make it easy to test within the context of a new or unproven circuit or P.C.B. The general philosphy is to get the memory interface working and then to use the DIR and DOL/DOH registers to help find any problems. An oscilloscope is also needed to check signals like the CLOCK , GO and for checking the operation of the various busses.

VI.1. The Memory Interface

The best way to check the function of the memory interface is to write and read values to either or both banks of coefficient registers. The correct operation of these is independant of the CLOCK , RESET settings or the contents of the SCR, ACR or TCR. The values that are written at this time are not important but writing 10101... and 01010... patterns will help locate any short or open circuits on the board. If little activity is observed from the IMS A100 then use the oscilloscope to check $\overline{CS}$, $\overline{CE}$, $\overline{W}$, ADR[0-6] and DATA[0-15] lines, and ensure the presence and correct timing of these signals. The best way to do this is to write a program on the host processor that loops continuously doing writes and reads to one IMSA100. These tests may not identify crossed data or address lines.

It is worth correcting any problems in this area before proceeding to the next series of tests.

VI.2. Clock, GO and output ready

Before checking these basic functions it is worth reseting the IMSA100 devices either by using the RESET pin or by powering down the system. This

is to ensure that earlier attempts to debug the memory interface have not accidentally written values to the SCR, TCR and ACR registers. The clock should be checked using an oscilloscope.

Problems with impedance mismatching may cause excessive voltage overshoot and/or undershoot, or cause the clock to not meet the specification in some other way through excessive ringing. Lack of drive in the clock driver will cause poor '0' and/or '1' levels. If the clock does not quite meet the specification but is present and is a reasonable shape, it is probably worth leaving the problem until the rest of the system has been debugged.

With the clock running either pull GO high or provide a series of GO pulses. If an IMSA100 is to be used as a master simply pull GO high with a resistor for this test since all the devices are still set as slaves. Under these conditions the OUTRDY pin should be providing pulses.

VI.3. Setting up SCR values

Before proceeding, the SCR registers should be set to #002 in slave IMSA100 devices or #003 in any master.

VI.4. Checking the data path

The next step is to check the data path from the input of the first IMSA100 to the output of the last one. It is worth writing a program to display the contents of the DOL/DOH registers on a monitor or TV screen.

All coefficients should be set to '0' together with CascadeIn[0..11] of the first device. A GO signal is now needed which is provided by the support logic or by writing to the DIR of an IMSA100. The method depends on the system under test, which will either be a master generated GO or externally generated GO . The value written to a master IMSA100 should not effect either its output or the contents of its DIR/DOL regisers, since all the coefficients are set to '0'. If there is the option of placing a value on CascadeIn[0..11] of the IMSA100, that value should appear in the DOL/DOH register. For a single IMSA100 the value will be delayed by 32 cycles of GO, and for many devices the delay will be a multiple of 32 cycles.

The following tests are valid for all coefficient word lengths, although only the least significant 4 bits will be used. The source of the GO signal is unimportant and the devices can be set for fast or normal output mode. However, SCR[4-5] should be set to '0' to select the [7-30] field, and the answers will then be the same as those given below. The values of coefficients and data and expected are given as hexadecimal numbers.

Set CascadeIn[0..11] on first device to	#000000
Set DIR on all devices to	#1002
Set all active coefficients to	#0004

If a master IMSA100 is used, continuously write #1002 to it's DIR register. Otherwise, apply a continuous or frequent GO signal, while writing data to the DataIn[0..15] port of all the IMSA100 devices. For the first 32 cycles of every device the result in the DOL/DOH register should increase linearly, in steps of #4008. The result will be split between the DOL and DOH register so that for the whole result #4008 DOH = #0004 and DOL = #0008.

1st cycle in the cascade	DOH=#0000	DOL=#4008
2nd cycle in the cascade	DOH=#0000	DOL=#8010
3rd cycle in the cascade	DOH=#0000	DOL=#B018
4th cycle in the cascade	DOH=#0001	DOL=#0020
5th cycle in the cascade	DOH=#0001	DOL=#4028
7th cycle in the cascade	DOH=#0010	DOL=#8030
7th cycle in the cascade	DOH=#0010	DOL=#B038

This test can be repeated using the DataIn[0..15] port instead of the DIR registers by setting the SCR values in all slave IMSA100 devices to #000. If the GO signal is generated from a master IMSA100 it will still be necessary to write #1002 to the DIR register repeatedly. The input value #1002 can be changed to other values to check other bits in the data path. Once the cascade path has been filled, the answers should be stable, and any variation indicates a problem somewhere.

Now that the devices have been exercised, it should be possible to remove any error indication from the $\overline{\text{ERROR}}$ pin by writing '0s' to ACR[1-2] followed by writing '1s' to arm the register.

The above tests only check the memory interface and the data path through the IMSA100 devices. However, with these working, the debugging of the rest of system is made easier. Once all this works most of the system will be fully functional.

VI.5. Fault finding guide

It is assumed throughout that power has been applied to all the V_{CC} and GND pins correctly. If it has not, expect very unpredictable behaviour and/or possible damage to the devices.

When a problem is encountered it is often worth varying the power supply voltage or changing the clock frequency. This will often indicate the nature of the problem by showing if it is due to timing or perhaps noise. The following checks may also help to diagnose the problem.

- If there is response from the memory interface check the following:
  - $\overline{CS}$ is low (when it matters)
  - $\overline{CE}$is pulsing
  - The addresses are valid
  - $\overline{W}$ is working
  - Any memory bidirectional data buffers are working in the right direction.
- If there is no GO signal from a master IMSA100 check the following. The clock has to be present and the RESET pin high before the SCR, ACR or TCR registers can be written to.
  - There is only ONE master.
  - There are no shorts on the GO track.
  - SCR[0] is set to '1'.
  - TCR is set to all '0s'.
  - The clock is present.
  - RESET is high.
- If there is no OUTRDY signal check the following.
  - GO signals are present on some rising clock edges.
  - TCR is set to all '0s'.
  - There are no shorts on the OUTRDY track.
  - RESET is high.
- The answers are wrong, which could be almost anything. However, the following checklist should diagnose the problem.
  - The SCR registers are set to the correct value.
  - The ACR registers are set to the correct value.
  - The TCR registers are set to all zeros.
  - All IMS A100 devices are in the same output mode. (SCR[10])
  - There is only one master. (SCR[0])
  - The output word selection is sensible. (SCR[4-5])
  - The data input source is correct. (SCR[1])
  - The coefficients word lengths are right. (SCR[8-9])
  - The coefficients are stored in the right bank.
  - DOL and DOH are read in the right order.
  - The memory data and address lines are in the right order.
  - OUTRDY is not inverted wrongly in any external logic.
  - 4, 8 and 12 bit coefficients are written into the least significant bits of the 16 bit wide coefficient registers. Input data is valid when sampled.
  - The order of the coefficients has not been mangled by the memory map.

IMAGE PROCESSING WITH THE IMSA100

SUMMARY

I. INTRODUCTION

I.1. The aims of this document

The IMSA100 performance makes the real time processing of digital images a practical possibility. This document is a practical guide, which explains how the device is used to process digital images. The processing done by the IMSA100 will be some form of feature extraction, such as line, corner or edge detection. Feature extraction is often the first stage in the analysis of an image. Further analysis of an image, for example, deciding that a group of features in an image is a vehicle number plate, is a higher level function, beyond the scope of this document. This application note describes the following.

- The operations of filtering and edge detection of a picture or image using a technique of 2-dimensional convolution are explained. Some simple filter types including edge detection and contrast enhancement are described.
- The use of the IMSA100 to perform the 2 dimensional convolution, in order to process an image is described. This shows the simplicity of use of the IMSA100 in this particular application.

- The estimation of performance and cost, for processing an image using the IMSA100 is described. Several possible systems consisting of IMSA100 devices are given, to illustrate how easily the cost and performance may be controlled, by using different numbers of devices, and by altering the complexity of the system.
- The processing of images at real time speeds (20 frames per second) is described, and a hardware implementation of this is given. This shows the high performance possible using the device.

I.2. Document structure

The remainder of section I gives an introduction to signal processing, and shows the position of the IMSA100 within the field of signal processing, and more specifically its capabilities for the processing of digital images.

Section II gives gives a practical explanation of some of the concepts of image processing. Included is an explanation of how filtering and edge detection of a picture operates, and how this may be applied to the IMSA100.

Section III gives two possible systems which may be constructed using the IMSA100, from a medium performance system to a very high performance system which will operate at real time speeds. Included in this section is a description of how the performance of a prospective system may be estimated by trading off performance, complexity and cost.

Section IV concludes and summarises the findings of this application note.

I.3. An overview of signal processing

Signal processing is an area of engineering which fills many people with dread. This is not entirely surprising when one considers both the theoretical and practical aspects of the subject. On the one side there are the mathematical algorithms required to solve even the simplest problem. This has long been regarded as the territory of academics and not to be tackled by the average engineer. On the other side there is the circuitry required to implement these algorithms. Historically, systems have often required many complex circuits, with system design requiring a knowledge of analogue design, and also, in the more recent past, digital design.

Not surprisingly, there are very few scientists in the world with the knowledge or experience required to deal with all the aspects of signal processing design. Signal processing design now covers both analogue and digital design from the low end audio spectrum (40kHz) through the video spectrum (100MHz) to the top end of the radio spectrum (100GHz). When signal processing in all these areas began the techniques used were purely analogue. The power of digital signal processing now approaches the top end of the video spectrum. Although it is not yet possible to process pictures the size of a TV screen in real time, it will undoubtedly become possible within the next decade. One of the main applications of the IMSA100, as described in this document, is the processing of pictures in real time.

In the radio frequency (RF) spectrum, specialised devices are used as the first stage processing elements. These devices use components such as wave-guides, to give the necessary processing bandwidth (GHz). The fastest devices use materials such as Gallium Arsenide, often supercooled to improve its performance. However, these devices are expensive and their use is avoided if possible. The information extracted by these devices from a signal may be used by todays digital devices operating at speeds approaching 100MHz. In the future, todays digital devices may improve to a level where they encroach on the radio spectrum. However, it is likely that RF devices will always be required as the front-end processing elements at these high frequencies. The reason for this may remain that it is impossible to either sample or synthesise an analogue signal at speeds in excess of 100MHz, without resort to cost prohibitive technology.

I.4. Analogue and digital conversion

Signal processing techniques in both the Audio and Radio spectrum are advancing both theoretically with the development of new algorithms, and practically with the increase in the level of integration of integrated circuits. Wherever possible, the new levels of integration in conjunction with efficient digital algorithms are used, so that problems which were previously solved using analogue design are now solved using digital design.

Of course, it is nearly always necessary to communicate with the real world using analogue signals, so analogue to digital (A-D) and digital to analogue (D-A) converters are a necessity. This is why so much work is done to increase the speed and accuracy of the conversion which must ulti-

mately limit the speed of the complete system.

The current range of A-D and D-A converters on the market can sample at up to 100MHz. As might be expected, the limiting speed depends very much on the required accuracy, with slower conversion required to get improved accuracy. Of course, there is little point being able to do digital processing faster than the analogue conversion devices, so that in practice, the performance of conversion devices and digital processing devices proceed together.

So there are fundamentally two problems which hinder DSP development, one is analogue/digital conversion and the other is the digital signal processing itself.

I.5. Techniques for digital signal processing (DSP)

Digital signal processing has advanced rapidly since the major semiconductor manufacturers started to tackle the problem. Since then they have attempted to cram more and more raw processing power onto a single chip. At the same time they have realised that the signal processing devices need to be integrated into an entire system. So, they have devised families of devices which, however, require some considerable expertise to use. This evolution of devices has split into two directions.

The first approach is the more complex and achieves the best performance. It often involves hardware design which is not trivial, and the systems generated will generally only perform one task. Any slight change to the task (algorithm) may require a complete system redesign, which is both lengthy and expensive. However, the performance of these so called bit-slice machines has been and still is very high and has a permanent place in the field. Bit-slice machines use dedicated multipliers, accumulators and address sequencers often with several address and data bus paths to achieve high speed.

The second approach is simpler, and more versatile. However, the performance is considerably lower than the bit-slice engines previously described. Design involves using a general purpose processor (CPU) which has dedicated instructions to perform reasonably fast multiply, divide, add and subtract operations. The CPU does this by having dedicate parallel multipliers and barrel shifters integrated on the chip. The performance limit is not so much the on-chip operation as the time required to get the data off and on chip (memory bandwidth). Possibly the best known example of a signal processing CPU is the TMS 32010[1] ([1]TMS is a trademark of Texas Instruments) and its derivatives the TMS 32020 and TMS 32030.

The previous two approaches provide solutions to a large number of signal processing problems. However, one must accept either the performance limitations of the general purpose processor or the complexities of bit-slice design. In both cases the problem is bandwidth into the basic processing element. The fundamental limit is the rate at which memory can be accessed rather than the performance of the processing element itself. If the processing performed by the basic processing element can be increased and the required memory bandwidth can be reduced, an improved performance will be immediate. The IMSA100 uses a novel architecture to achieve these aims.

The IMSA100 is a processing element with considerable processing power, yet having an interface with moderate bandwidth requirement. This is achieved by having data storage on chip, processing the data in parallel, and storing the intermediate results of calculations. The IMSA100 has also been designed to accommodate many of the commonest DSP algorithms; including the discrete Fourier transform [2}, correlation and convolution [3] , and digital filtering [1]

I.6. Overview of image processing with the IMSA100

The IMSA100 is a digital processing device at the forefront of digital signal processing performance. It is capable of processing video bandwidth signals, as well as many other types of high bandwidth signals. The maximum input sampling rate of the IMSA100 is 10MHz, which means that it could, for example, process a [512 × 512] image at a rate of 40 frames per second. The device operates on digital data with a width of 16 bits, and will perform 80 million multiply accumulate operations per second (80 MOPS) a performance well in excess of most bit-slice machines.

The IMSA100 will perform calculations on signed 16-bit integers without any loss of accuracy or overflow, perform rounding correctly, and will also perform complex number processing [4] without any additional hardware. This makes it an extremely simple device to use in a wide variety of applications, as it deals with so many of the problems which have historically plagued signal pro-

cessing design. Immense care has been taken to ensure that the device is simple to use, for example, the microprocessor interface, which can be interfaced very simply with almost any industry standard processor.

Probably the most important aspect of the IMSA100 is that several can be used in parallel, with almost no 'glue' logic. In principle, there is no limit to the number and a system with 30 devices on a single board has been shown to work well. The processing of large images at high speed requires vast processing performance, making the IMSA100 capability of being able to use many devices in parallel absolutely invaluable.

II. PRACTICAL METHODS OF 2 DIMENSIONAL CONVOLUTION

II.1. dimensional convolution

The process of 2-dimensional convolution of an image is the action of comparing a reference template with a group of pixels, at every pixel point on an image. For example, if a [3 × 3] template were compared at every point on an image of size [5 × 5], there would be 9 valid comparison points as shown in Figure 1. The first of these valid comparisons surrounds pixel 1, the second pixel 2, and so on. The comparison is done in practice by a number of multiply and add operations. Consider the example with the first row being compared with the template. The result of the [3 × 3] convolution for the first 3 positions will be

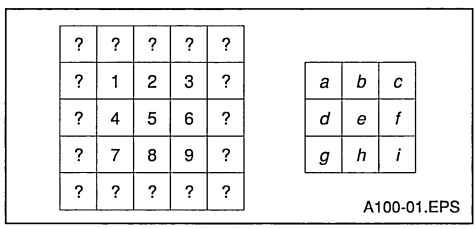
1 a.? + b.? + c.? + d.? + e.1 + f.2 + g.? + h.4 + i.5

2 a.? + b.? + c.? + d.1 + e.2 + f.3 + g.4 + h.5 + i.6

3 a.? + b.? + c.? + d.2 + e.3 + f.? + g.5 + h.6 + i.?

which is a total of 9 multiply-accumulate operations for every pixel in the image. The magnitude of the image data and the magnitude and sign of the template elements determine the type of features which will be extracted from the image. Some simple templates are described later in this section.

Figure 1 : [3 x 3] convolution on a [5 x 5] Image



In a real image, the magnitude of the pixels which is a measure of their blackness, is referred to as grey scale, having typically 8 bit accuracy. The alternative, which uses a single bit for each pixel, was used in the past, before digital grey scale processing was possible. Future picture processing will undoubtedly be capable of processing colour images. This is a complex field, little understood at the present time, outside the scope of this application note.

With grey scale images it is important that the result of any image transformation yields grey scale values within the limits of the original image. This being so, the sign and magnitude of the elements of the template must be chosen with care. It may be necessary to scale and/or invert the results of an image transformation, so that the resultant image can be observed in a normal grey scale.

It is usual for the template to be square, although it may be rectangular, and of any size. It is also normal when scanning a real image to traverse the picture as shown in the diagram, i.e. traversing a row, moving down, traversing again and so on until the entire image is scanned.

One point of interest concerns the outermost pixels, which represent invalid data. For a [3 × 3] template a perimeter of one pixel width is invalid, for a [5 × 5] template the outermost 2 pixels are invalid and this redundancy increases as template sizes increase. This does not matter much for large image sizes, but must be borne in mind if large templates, with small images are being used. For the remainder of this section edge effects will, for convenience, be ignored.

II.2. Convolution template types

II.2.a. LOW PASS FILTER

The effect shown in Figure 2, is of a low pass filter. The numbers have been chosen to show the smoothing effect of the filter. Notice that this is indeed a low pass filter, and that the pixel values are changing at a frequency which is approximately the cut-off frequency of the filter. The filter has effectively changed a black and white image into a blurred grey image.

If this convolution is regarded as part of a picture with a pixel rate of 5MHz the cut-off frequency, above which all frequencies are removed, would be 2.5MHz. (The figure of 5MHz has been chosen as it is the rate at which an IMSA100 can process the data.) The cut-off frequency can be reduced by making the reference template (convolution kernel)

larger. For example a [9 × 9] convolution kernel would have a cut-off frequency of 870KHz.

For the low pass filter kernel no sign modification or scaling of the final image is necessary. Only when the result is outside grey scale limits will any modification be required.

II.2.b. EDGE DETECTION

Edge detection is illustrated below with a Sobel operator. This operator combines a vertical and a horizontal edge detector into a single Sobel operator as shown in Figure 3.

It may be observed that the effect of applying a horizontal edge detection to an image, followed by applying a vertical edge detection to an image, and summing the results, will be exactly the same as directly applying the sobel operator to the image. This same principle of adding operators together, may be applied to many different operators with some interesting results. It is not within the scope of this application note to investigate this subject further.

The following operations, shown in Figure 4, on part of an image illustrate the effect of the Sobel operator. It is possible to obtain similar results, by doing a vertical and a horizontal edge detection, squaring and adding the results, and taking the square root to give a result for each final pixel. This is the ideal edge detector, but the cost of squaring twice and a square root is often cost prohibitive, with the Sobel operator a very satisfactory alternative.

Figure 2 : Illustration of Low Pass Filter

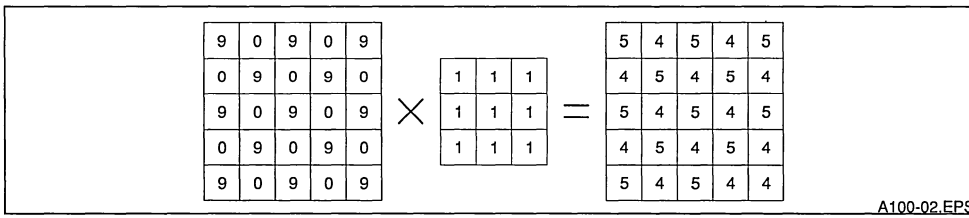


Figure 3 : Sobel Operator Formation

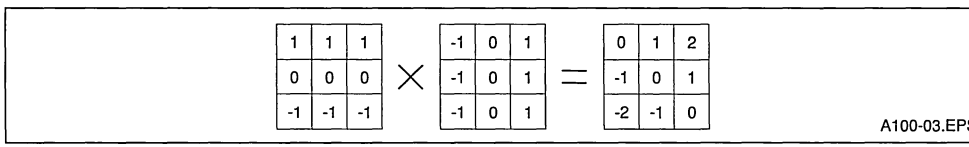


Figure 4 : Illustration of Edge Detection

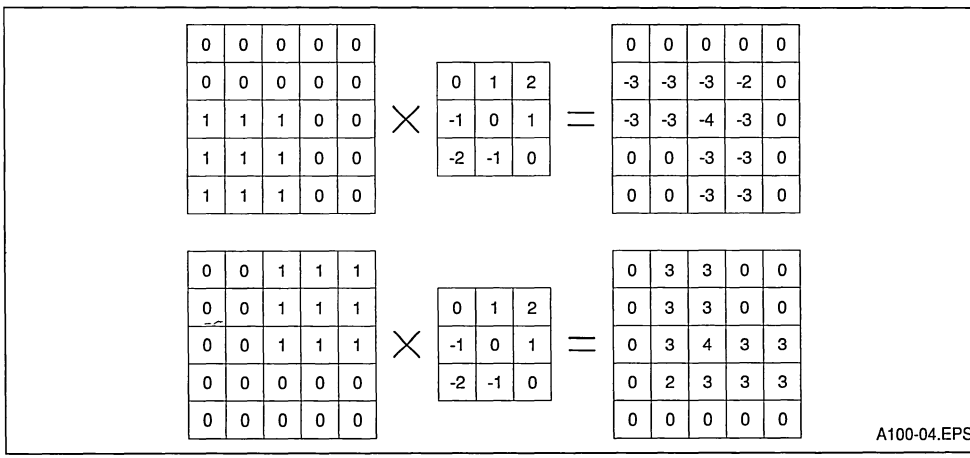


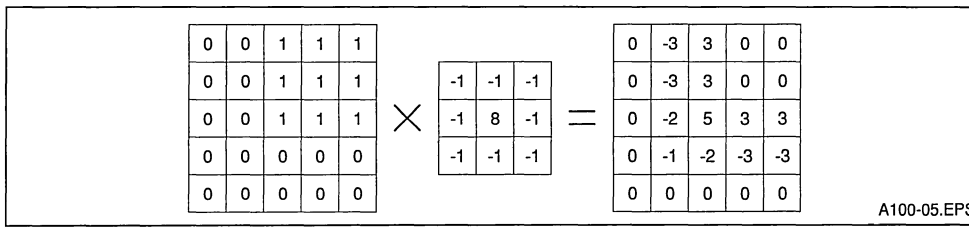
Figure 4 shows the result of 2 convolution kernels operating on the different images. This illustrates the requirement for scaling and sign inversion. Before the resultant images can be displayed all negative numbers must be sign inverted and a scaling factor of 4 must also be applied. It is interesting to note that the reason for the sign change is the direction of travel of the convolution kernel across an edge transition. Also, as will be shown later, the steepness of the edge transition is important.

II.2.c. LAPLACIAN FILTERING (edge detection)

Laplacian filtering uses a homogeneous operator, which means that it is the same in all directions. With the use of a Laplacian edge detection operator edges in all directions can be detected. This is different from the previous non-homogeneous edge detection (Sobel) operator, where edges in all directions except one can be detected.

The effect of the [3 × 3] Laplacian operator is shown in Figure 5. As the operator passes over an edge, the magnitude of the result increases and there is a sign change. Also, the original pixels will remain, in areas where there is no edge to be detected. In order to detect the edges, after the 2-D convolution has been done, a 3 stage process is necessary. First the result is scaled down by a factor of 9. Second, a full rectification is done, converting all negative to positive numbers. Third, the background information is thrown away, by introducing a suitable threshold below which the result is considered to be zero.

Figure 5 : Illustration of Laplacian Filter



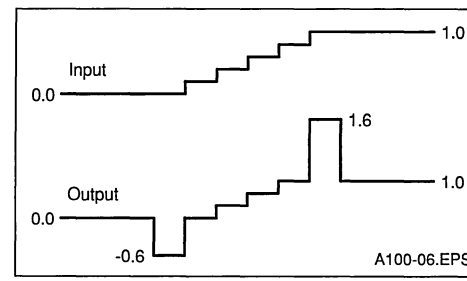
II.3. Effect of template size

The previous sections have shown the effect of several [3 × 3] convolution kernels. This kernel size is very effective in many applications, while requiring moderate processing for its calculation. One of the main reasons for not using larger templates is that the processing requirement becomes excessive, mainly due to the large number of multiply operations. The advantages of large kernels are twofold, firstly the larger kernels have a filtering effect which reduces the effect of noise, and secondly the larger kernels are able to detect gradual changes in brightness across a group of pixels.

It must be remembered that single bit pixels are not used, and the pixels are represented in grey scale with range from 0 to 255. This means that in a real image, edges will span several pixels, and to detect these edges will require a larger convolution kernel.

If, for example, a [3 × 3] kernel is used to detect an edge which changes from black to white linearly over 5 pixels, then the maximum and minimum resultant pixel is 1.6 and -0.6 respectively, as shown in Figure 6. Each pixel is represented by a single step. This result must be compared with the result in Figure 5, where an instantaneous change between 2 pixels gives an output of -3.0 and 3.0. The results of -0.6 and 1.6 are barely enough to detect an edge.

Figure 6 : Effect of Gradual Edges on Convolution



III. HARDWARE REQUIREMENTS FOR 2-D CONVOLUTION

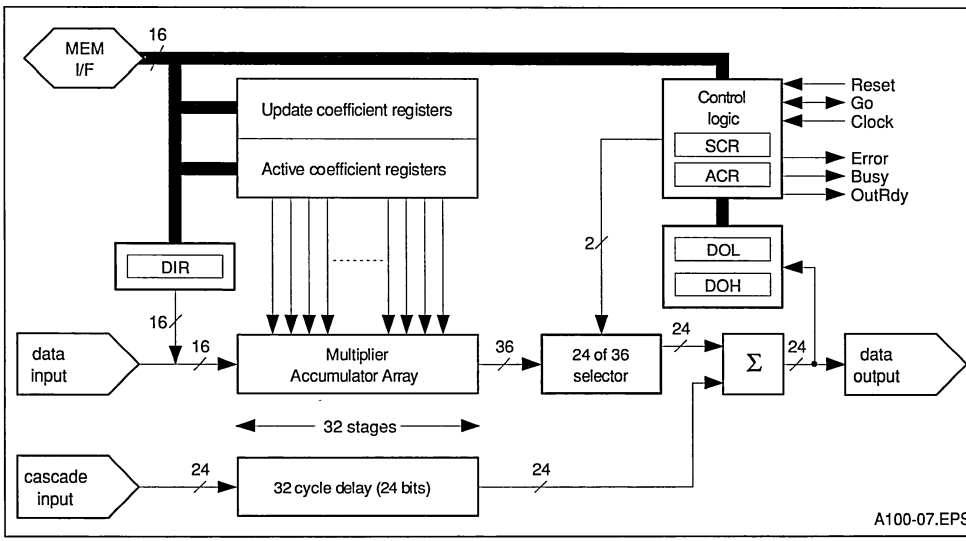
Two possible hardware implementations of 2-D convolution using the IMSA100 are described in this section. Because these two implementaions use exactly the same principle of operation, the IMSA100 device, which is common to both, will first be described. The fundamental difference between the two designs is as follows. In the lower performance system all image data is transferred to the IMSA100 across a comparatively slow memory interface. In the high performance system all image data uses the dedicated input and output ports of the IMSA100. These dedicated ports permit, with the addition of some dedicated hardware, a processing rate of 5 million pixels per second.

Implementation of 2-dimensional convolution on the IMSA100 involves loading the elements of the convolution kernel into the coefficient registers, and passing the entire image through the device while storing the resultant image. To obtain maximum throughput this should be a continuous operation, and will consist of a sequence of alternate read/write operations starting at the first pixel of the first row of the image and finishing with the last pixel of the last row of the image. The two fundamental problems are to arrange that first, the convolution kernel elements are loaded into the appropriate coefficient registers, and second that the pixel data is ordered correctly both before and after processing. The required initialisations of the IMSA100 are also described.

III.1. The IMSA100 model

The IMSA100 model is shown in Figure 7. The many component parts of the IMSA100 are included to add flexibility, so that many signal processing algorithms can be implemented. This means that the device can be used in many signal processing applications. The fundamental operation of the device is a high speed multiplier-accumulator, which functions as a pipeline of 32 multiplier-accumulator devices. The peripheral circuitry simplifies the use of the device.

Figure 7 : User's Model of the IMSA100



The essential elements of the device, in so far as they are important to this discussion of 2-D convolution, will now be described.

- The multiplier accumulator array is the powerhouse of the device. This is a 32 stage pipeline of multipliers, which multiply 32 elements of input data with the contents of the current coefficient registers in between 2 and 8 cycles. The cycle time is between 100ns and 400ns, and is a function of the coefficient width. There is no loss of accuracy in this section because all calculations are at 36 bit accuracy.
- **The data input** is either from the dedicated data input port or via the data input register (DIR). The DIR can be accessed from the memory interface port, which may be connected to a microprocessor. The fastest data access is by the direct input port, with input using the DIR register being usually 2 to 4 times slower.
- **The data output** is taken from the high speed multiplexed data output port or from the data output registers (DOL and DOH), which contain the 24 bits of output data. These registers, like the DIR register, will normally be accessed much slower than the direct data output port.
- **The coefficient registers** are used to store the convolution coefficients, for which only the current coefficient registers are required. Because there is no need to bank-swap coefficient and update registers, neither the update coefficients nor the bank swap capability are ever used in this application.
- **The cascade input** is used for simple connection of devices. The cascade input port is multiplexed in exactly the same way as the data output port, so that direct connection between the two and use of the GO signal for synchronisation are all that is required to cascade devices. For 2 dimensional convolution requirements, only one IMSA100 is required for doing a convolution with a kernel containing less than 33 elements, although more devices can be used for improved performance as will be shown later. Whenever more than one device is used the cascade input port is required.
- **The 32 cycle delay** element delays the cascade input data by exactly the same time as the multiplier accumulator array. It can be used in conjunction with the data input port to add together 2 streams of pipelined data. This is very useful, particularly for convolution requiring data partitioning. Real time 2-dimensional convolution of images using the IMSA100 requires the use of this delay element.
- **The control registers** are used to initialise the device, and for some of the working operations of the device. These are referred to as the Static control register (SCR) and the Active control register (ACR) respectively. The ACR can be altered during the operation of the device whereas the SCR cannot. The use of these registers as regards 2-dimensional convolution will be described later.
- **The output signals** are described adequately in the IMSA100 data sheet [6] and will not be described further here, except for the GO signal which is relevent to the discussion. The GO pins, of all the IMSA100 devices which are cascaded together, will be joined. GO is used for synchronisation of a cascade of devices, and is not needed in a system with only a single IMSA100, unless the cascade input of that device is used. GO is set up from the SCR to be either a master or slave, and there is never more than one master.
- **GO** is a special signal used to synchronise the cascade and data input ports. If the data input port **Din** or the cascade input port **Cin** is driven by external hardware, then all the IMSA100 devices will be set to slave mode and external hardware will be used to drive the GO pin. If neither the cascade input port or data input port are driven by external hardware, (when all data will use the memory interface) then one of the IMSA100 devices in the cascade will be configured as a master. The master which should be the **last** device in the cascade, drives the GO signal, and all the other devices synchronise their cascade and data inputs from the GO signal they receive. The GO signal master could in theory be driven by any of the devices in a cascade, and this would work for a short cascade. However, operation cannot be guaranteed, whereas an infinite length cascade will work if the master is the last device in the cascade.

III.2. IMS A100 initialisations for convolution

The following description summarised the initialisations of the IMSA100 devices which will be required, prior to the operation of 2-D convolution. A full understanding will require the use of the IMSA100 data sheet [5]. The settings necessary for a 2-dimensional convolution, using 8-bit grey scale data and 8 bit coefficients are described.

The coefficient size is set to 8 bits by setting bits 8

(=1) and 9 (=0) of the SCR. As 8 bit grey scale is used the top 8 data bits from either the data input port or DIR (each 16 bits wide) will be zero.

The result of the 8 by 8 multiplication will require 16 bits, and the 32 stages of accumulation will require a further 5 bits so that the final result, will require 21 bits accuracy. The result required is manipulated internally by a selector so as to be invisible to the user. The significant 8 bits of the result are obtained by setting bits 4 (=0) and 5 (=0) of the SCR, and reading data from the bottom 8 bits of the DOL register.

If there is a cascade of devices the lower 8 bits of output appear on the lowest 8 bits of the multiplexed data output port, which will be connected to the cascade input of the next device in the cascade. By this means scaling is done automatically, and is invisible to the user. The whole purpose of this is that many devices can be cascaded, and appear like a single device with a number of stages which is a multiple of 32.

The remaining SCR register settings are as follows. Bank swap mode will be set to off. Data mode will be set to either input data from the DIR register or data input port depending on the application. Fast output will be set to off for this application.

The ACR will not generally be needed for this application as no bank swapping between the active and update coefficient registers is necessary. It may be necessary to examine the selector overflow and cascade adder overflow bits of the ACR should an error occur (error pin goes low).

III.3. IMSA100 coefficient placement and data flow

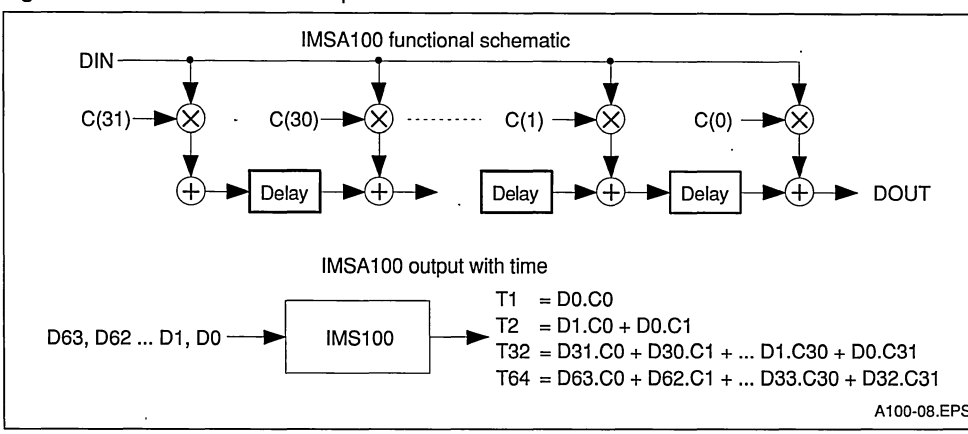
Section II.2 describes some of the convolution kernels which are used to perform feature extraction and filtering on an image. The following discussion describes how these kernel elements are mapped onto the coefficient registers of the IMSA100 so that 2-D convolution is performed.

The IMSA100 can be regarded as a 32 stage multiplier accumulator with 32 constant coefficients, which will be consecutively multiplied with a stream of incoming pixels. The current coefficients are labelled from C(0) to C(31) where C(0) is closest to the output, and C(31) is closest to the input, as shown in Figure 8.

In Figure 8, pixel data presented at the Din port (or DIN register) at time 0 is referred to as D0. Immediately after data is written, at time T1, a result will be read from the Dout port (or DOL/DOH register). For the first 32 cycles (T1 to T32) of the IMS A100, partial results for data D0 to D31, and coefficients C(0) to C(31) will be output from the device. The results at time T1 and T2 are given.

From T32 onwards the device presents full results at its output, and the result at time T32 and T64 are given to illustrate this. The steady state of the device yields the accumulation of 32 multiply operations which have taken place over the previous 32 cycles. Notice also that at any instant the machine contains 32 pieces of information (state), which are the 32 partially accumulated results, as they proceed through the 32 stage pipeline.

Figure 8 : Illustration of IMSA100 Pipilined Circulation



If there is a cascade of 2 devices, there are 64 coefficients which can be referred to as C[0] to C[63]. The output from the second device in the cascade is the sum of 64 multiply operations which have accumulated over the previous 64 cycles. This principle can be extended to many IMSA100 devices, so that long multiply-accumulation operations can be done. It is essential to be able to perform long cascades so that large convolutions are possible. For example, a 128 point convolution will require 4 IMSA100 devices in cascade.

This also applies to 2-Dimensional convolution. For instance, an [11 × 11] convolution using 121 stages, will require 4 IMSA100 devices. Of course, 7 stages are not required, which means that 7 of the coefficients (C[127] to C[121]) of the first IMS A100 in the cascade will be set to zero.

As can be recalled from section II, a [3 × 3] convolution requires the accumulation of 9 multiply operations. Similarly, a [5 × 5] convolution, illustrated in Figure 9, will require 25 stages of multiply-accumulation. The only problem is that the coefficients must be loaded in the correct coefficient locations, and the input and output data must be ordered correctly, so that the IMSA100 architecture can be utilised. This is described in the following section.

III.4. Image scanning for a microprocessor based system

The following description will normally only apply to a system using a memory interface, for the transfer of all data to and from the IMSA100. It is perfectly possible to use the following pixel sequencing operations, for transferring data to the IMSA100 devices across the high speed data input and output ports. However, this is not advised as the sequencing operations using normal hardware are complex, but are quite easy with a microprocessor. The additional hardware could be better used for implementing an extremely high performance system, such as described later.

Figure 9 : Pixel Scanning and Coefficient Ordering

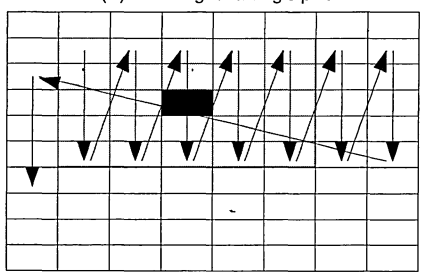
(A) Pixel array

D0	D10	D20	D30	D40	D50	D60	D70
D1	D11	D21	D31	D41	D51	D61	D71
D2	D12	D22	D32	D42	D52	D62	D72
D3	D13	D23	D33	D43	D53	D63	D73
D4	D14	D24	D34	D44	D54	D64	D74
D5	D15	D25	D35	D45	D55	D65	D75
D6	D16	D26	D36	D46	D56	D66	D76
D7	D17	D27	D37	D47	D57	D67	D77
D8	D18	D28	D38	D48	D58	D68	D78
D9	D19	D29	D39	D49	D59	D69	D79

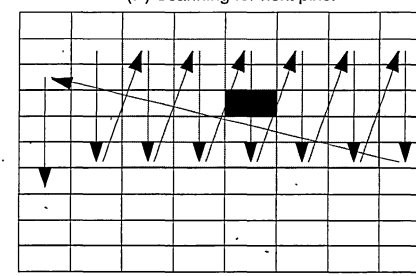
(B) IMSA100 coefficients

C24	C19	C14	C9	C4
C23	C18	C13	C8	C3
C22	C17	C12	C7	C2
C21	C16	C11	C6	C1
C20	C15	C10	C5	C0

(C) Scanning for a single pixel



(D) Scanning for next pixel



A100-09.EPS

III.4.a. IMAGE SCANNING FOR 2-D CONVOLUTION IMPLEMENTATION

A pixel array (A) with 10 rows and 8 columns is used purely for convenience. The convolution kernel with 25 coefficients is shown in (B). The order of the coefficients is critical, starting at the bottom right and proceeding one column at a time (Remember that C0 is the coefficient of the last stage of the cascade). The scanning pattern for the image is shown in (C) and (D). The dark black squares are valid output pixels, each of which represent the convolution of 25 pixels with 25 coefficients.

If the light grey area of pixels is written to **Din** as shown in (C) the order will be D11 then D12 and so on in columns until D55 is written. Immediately after D55 is written to **Din** a valid pixel is read from **Dout**. The value of this pixel will be

D33out = C0.D55 + C1.D54 + C2.D53 + ...
+ C23.D12 + C24.D11

After this D51 is written followed by D52, D53, D54, D55 after which another valid pixel can be read.

D34out = C0.D65 + C1.D64 + C2.D63 + ...
+ C23.D22 + C24.D21

In other words, for every 5 pixels written one valid pixel is read, from the beginning until the end of the row. At the end of the row go back to the start of the row and move down a row repeating until the entire image is scanned. The net effect is a completely convolved image. This is inefficient as the entire image is effectively written to the IMSA100 **FIVE** times.

There is fortunately an optimisation which can be incorporated. The principle is that the some of the IMSA100 coefficients are set to zero, so that those stages act only to store and delay accumulated results. This is described in the following section.

III.4.b. IMPROVED IMAGE SCANNING FOR 2-D CONVOLUTION

Improved performance can be obtained by modifying the previous image scanning technique. The improvement is obtained not by processing the individual pixels faster, but by passing the pixels through the IMSA100 fewer times. This is illustrated in Figure 10.

Figure 10 : Pixel Scanning and Coefficient Ordering - High Performance

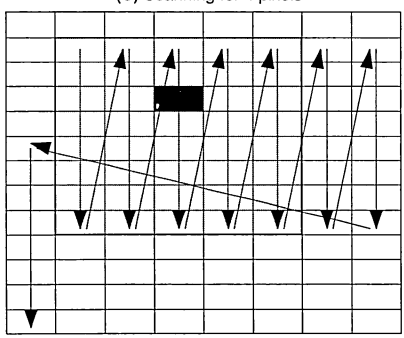
(A) Pixel array

D0	D10	D20	D30	D40	D50	D60	D70
D1	D11	D21	D31	D41	D51	D61	D71
D2	D12	D22	D32	D42	D52	D62	D72
D3	D13	D23	D33	D43	D53	D63	D73
D4	D14	D24	D34	D44	D54	D64	D74
D5	D15	D25	D35	D45	D55	D65	D75
D6	D16	D26	D36	D46	D56	D66	D76
D7	D17	D27	D37	D47	D57	D67	D77
D8	D18	D28	D38	D48	D58	D68	D78
D9	D19	D29	D39	D49	D59	D69	D79

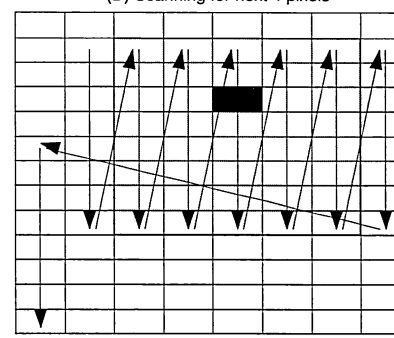
(B) IMSA100 coefficients

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
C36	C28	C20	C12	C4
C35	C27	C19	C11	C3
C34	C26	C18	C10	C2
C33	C25	C17	C9	C1
C32	C24	C16	C8	C0

(C) Scanning for 4 pixels



(D) Scanning for next 4 pixels



A100-10.EPS

Data is written to the IMSA100 devices as before except this time data is scanning over 8 rows at a time, starting at D11 and finishing at D58. Scanning over the fifth column of 8 pixels from D51 to D58 will yield 4 valid pixels, starting at the black square.

D33 out = C0.D55 + C1.D54 + C2.D53 + ...
+ C23.D12 + C24.D11

D34 out = C0.D56 + C1.D55 + C2.D54 + ...
+ C23.D13 + C24.D12

D35 out = C0.D57 + C1.D56 + C2.D55 + ...
+ C23.D14 + C24.D13

D36 out = C0.D58 + C1.D57 + C2.D56 + ...
+ C23.D15 + C24.D14

Immediately after D33out is read from **Dout**, D56 is written to **Din**. The partially accumulated result for pixel D43out is then stored in the empty slot (The empty slot is the position in the IMSA100 which would accumulate C5 with the data at **Din**. As this coefficient is set to zero no accumulation takes place, and this stage acts as delay and storage of data only) The next pixels D56 and D57 are written, and the partially accumulated result for D44out and D45out are then stored in the IMSA100 pipeline. At this time there will be 3 partially accumulated results D43out, D44out and D45out, which will be required for processing the next column.

At the end of the column scan, after the pixel D58 is written, D61 followed by the remainder of that column of 8 pixels, are written. This yields a further 4 pixels as given below.

D43out = C0.D65 + C1.D64 + C2.D63 + ...
+ C23.D22 + C24.D21

D44out = C0.D66 + C1.D65 + C2.D64 + ...
+ C23.D23 + C24.D22

D45out = C0.D67 + C1.D66 + C2.D65 + ...
+ C23.D24 + C24.D23

D46out = C0.D68 + C1.D67 + C2.D66 + ...
+ C23.D25 + C24.D24

The scanning technique which scans across 8 rows at a time, while 4 rows of pixels are written, is 2.5 times more efficient than the previous technique, where only 5 rows are written, for a single output row. It is simple to calculate the efficiency for any number of zeros inserted into the coefficients of the IMSA100.

III.4.c. CONVOLUTION EFFICIENCY

For any system, there will be a fundamental pixel processing rate. As shown in section III.5 the processing rate, writing pixels across a high performance memory interface is unlikely to be better than 4 Mpixels per second. Realistically 2Mpixels per second is a more probable performance. However, as shown above, there will be an efficiency factor, dependent upon the the convolution technique used, which will reduce the performance further. The best that can be done uses an image scanning pattern as shown in Figure 10.

To calculate the efficiency, the number of stages and the number of zeros must be known. These calculations assume that the maximum number of zeros will be used, for whatever number of A100s are selected.

$$\text{stages} := \text{number.of.A100s} \times 32 \quad (1)$$

$$\text{zeros} := \frac{\text{stages}}{(\text{filter.size})^2}\ \text{DIV}(\text{filter.size}-1) \quad (2)$$

$$\text{Efficiency:} = \frac{\text{zeros}+1}{\text{zeros}+\text{filter.size}} \quad (3)$$

As a simple example, it is known to take 500ns to process a single pixel, and the efficiency is calculated at 50%. The expected processing rate will be 1Mpixels per sec.

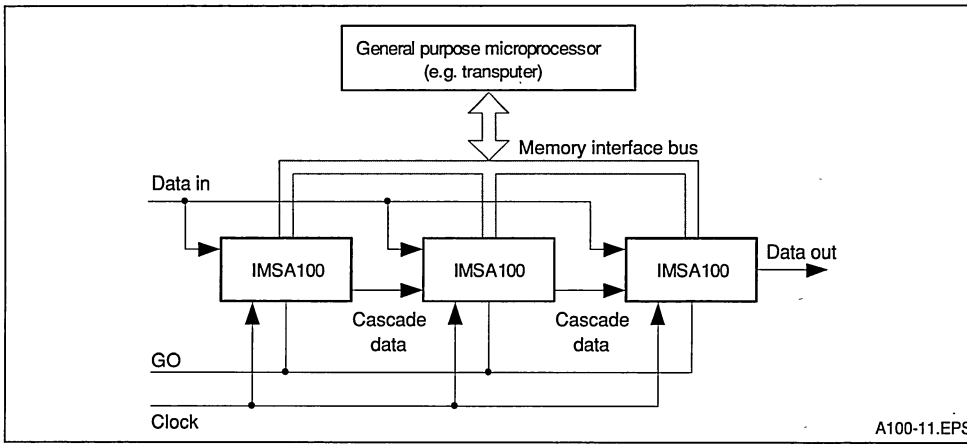
There is an obvious trade off between the number of A100 devices used and efficiency. A small number of zeros increases efficiency greatly. However, as efficiency approaches 100% the added cost of more IMSA100 devices, to give more stages, will not give a proportionate increase in performance. No figures are given here, as it is simple to calculate the numbers, for any given application.

III.5. Moderate speed image convolution

A moderate image convolution rate can be obtained by using a very simple design incorporating an 8 or 16 bit processor, which controls one or several IMSA100 devices. A typical system is shown in Figure 11. The system chosen uses an extremely high performance 16 bit processor, the IMST212. The limiting speed of this system is either the rate at which data can be transferred across the IMSA100/IMST212 memory interface, or the rate at which data can be transferred to and from an external system. The external system may consist of camera, frame grabbing hardware and some form of image displaying capability. For the purpose of argument it will be assumed that the IMSA100/IMST212 memory interface is the limiting factor.

The performance of this system may be easily estimated, for whatever processor is used. This estimation assumes that the image resides in memory before processing starts, and that the data input port is not used. The resultant image will also reside in memory after processing.

Figure 11



The processing of a single pixel involves 5 steps which are in order

read from memory	= 100ns
write to IMS A100	= 100ns
IMS A100 processing,	= 200ns
read result from IMSA100	= 100ns
write result to memory	= 100ns

This shows that the time to process a single pixel will be 600ns so that for a complete picture of size [512 × 512] a processing rate of 6 frames per second may be possible. While this may be achievable in theory there are several problems which lead to a reduction in this performance.

- The data must be transferred both to and from the system,, before and after processing. In practice this may take longer than the processing itself.
- The data must be read and then written by the processor, usually into an internal register, which will consume at least 2 extra cycles (100ns minimum)
- The data accessed by the processor will be in the form of a 2 dimensional array of pixels. The processor has to calculate the array subscripts for every pixel in the image, which will consume at least 4 processor cycles (200ns). The order with which pixels are loaded is not simply row by row and is described elsewhere. (section 3.4)
- The nature of the convolution algorithm means that the image may need to be split up into small blocks, which must be overlapped, to give a continuous convolution of the entire screen. This will result in an inefficiency of between 10% and 50% depending on the size of the blocks and the degree of overlap.
- The algorithm implemented on the IMSA100 has a fundamental efficiency as described in section 3.4. Equations are given for the calculation of this efficiency. The algorithm should be arranged so that the efficiency is better than 50%.

The effect of all these factors that a simple microprocessor based system is likely to have a processing rate of between 1 and 4 frames per second. Even this will require an high performance processor with optimised software.

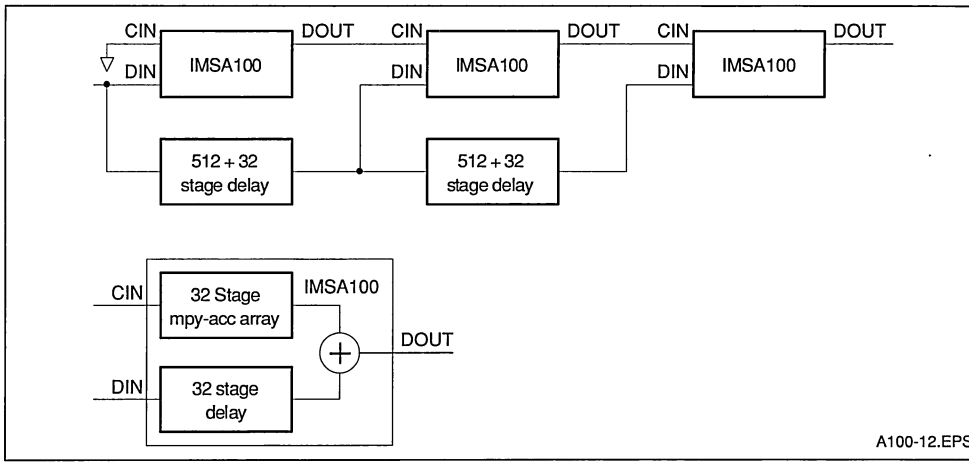
III.6. Very high speed image convlution

Faster speeds require a slightly altered algorithm and dedicated hardware. The following describes a hardware implementation capable of processing speeds up to 20 frames per second. The hardware setup is shown in Figure 12. This figure illustrates the hardware required to perform a [3 × 3] convolution on a [512 × 512] image. Larger convolutions on larger images are possible with the addition of extra hardware. For example, a [31 × 31] image convolution could be done using 31 IMSA100's and 30 sets of shift registers. Each shift register has a 512 + 32 stage delay.

In the example shown, 2 rows of data are stored in long shift registers, while the third row enters the data input port of the first IMSA100 in the cascade. The first IMSA100 in the cascade has its cascade inputs grounded, while all other IMSA100 devices have their cascade inputs connected to the data outputs of the previous IMSA100 in the cascade.

The arrangement ensures that 2 pixels one above the other in an image are fed simultaneously into the cascade input and data input of each IMSA100.

Figure 12



The IMSA100 devices will process one line of pixels at a time, and the entire image will eventually pass through every IMSA100. Because the system is fully pipelined this results in no performance degredation.

Each stage of the cascade requires an IMSA100 and a long shift register. The IMSA100 is like a 32 stage delay element, where the cascade input delayed by 32 stages is added to the data input after it has been through a 32 stage multiplier-accumulator array. The 32 stage delay of the IMSA100 means that the shift register delay must be a single line delay (512 pixels in this case) plus 32 stages.

The data throughput rate depends on the coefficient size selected for the IMSA100's, and is unaffected by the number of stages. If 8 bit pixels with 8 bit coefficients are selected, a data rate of 5kHz is achieved. For a [512 × 512] image this gives a convolution time of one frame in 50mS. This is a frame rate of 20Hz with faster speeds achieved by selecting regions of interest or multiplexing frames between several boards.

As a further example application, it is required to pass a [31 × 31] pixel kernel template over a [1024 × 1024] image at 50 frames per second. Processing will require 310 IMSA100 devices, segmented over several boards. One configuration would use 30 [1024 + 32] delay stages, and would require 31 IMSA100's to process the image in 200 ms. Therefore 10 such boards would be required, and a means of multiplexing the individual images at a bandwidth of 50 Mpixels per second. This is mainly a problem of data distribution. The processing problem has been solved by the capability of the IMSA100.

IV. CONCLUSIONS

This application note has shown how the IMSA100 may be used to perform fast processing of digital images. Some typical image processing functions have been described, including edge detection and filtering of a picture.

The versatility of the IMSA100 has been shown by the following observations.

- The IMSA100 is capable of processing images at real time speeds, 20 frames per second, and a hardware implementation of this has been described. This processing rate is possible because of the high speed sampling rate of 10 million pixels per second of the IMSA100.
- It is simple to build a lower performance system consisting of several IMSA100s and a microprocessor, with a capability of processing at between 2 and 4 frames per second. The simplicity of the microprocessor interface which turns the device into a memory mapped peripheral helps to achieve this.

- Identical hardware implementations can be used for different sizes and types of 2-Dimensional filters. It is therefore not difficult to modify the signal processing algorithm after the hardware design is complete. This is eased by the general capabilities of the IMSA100 for many signal processimg applications, involving the implementation of specific algorithms on the general purpose architecture.
- Even after hardware design is complete it is possible to increase or reduce performance as necessary, simply by increasing or reducing the number of IMSA100 devices in the system. This is only possible because the device is designed specifically so that several may be used in parallel.

VII. REFERENCES

1 Digital filtering with the IMS A100, Application note 1, Hossein Yassaie, INMOS Limited, Bristol.

2 Discrete Fourier transform with the IMS A100, Application note 2, Hossein Yassaie, INMOS Limited, Bristol.

3 Correlation and convolution with the IMS A100, Application note 3, Hossein Yassaie, INMOS Limited, Bristol.

4 Complex (I & Q) processing with the IMS A100, Application note 4, Hossein Yassaie, INMOS Limited, Bristol.

5 IMS D703 reference manual, INMOS Limited, Bristol.

6 IMS A100 data sheet, INMOS Limited, Bristol.

7 Index mappings for multidimensional formulation of the DFT and convolution, Burrus C.S., IEEE Trans. on ASSP, 25:239--242, June 1977.

8 DFT/FFT and convolution algorithms -- theory and implementation, Burrus C.S. and Parks T.W., IEEE Trans. on ASSP, 25:239-242, June 1977.

SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

CASCADING IMSA110s

1. INTRODUCTION

The IMS A110 is a single-chip programmable and cascadable device suitable for many high speed image and signal processing applications. It consists of a configurable array of multiply-accumulators (420 MOPs), three programmable length 1120 tage shift registers, a versatile post-processing unit and a microprocessor interface for configuration and control purposes. The comprehensive on-chip facilities makes a single device capable of dealing with many image processing operations. A simplified block diagram is shown in Figure 1.

For some applications however, the power and versatility of a single IMS A110 is not sufficient, in these cases a cascade of devices often provides a solution. The purpose of this document is to describe some of the most useful ways to cascade IMS A110s to achieve even higher performance and as such does not cover the use of the backend processor or device applications.

2. OPERATION OF A SINGLE IMS A110

The A110 may be set up as either a one or two dimensional multiplier accumulator array (MAC).

2.1 One dimensional operation of an IMS A110

For one dimensional operation the first delay PSRc is set to some arbitrary value (normally zero) while PSRb and PSRa are set to zero. N.B. at any given point in time the first MAC stage in bank c is

processing the oldest data while the last MAC stage of bank a is processing the newest data.

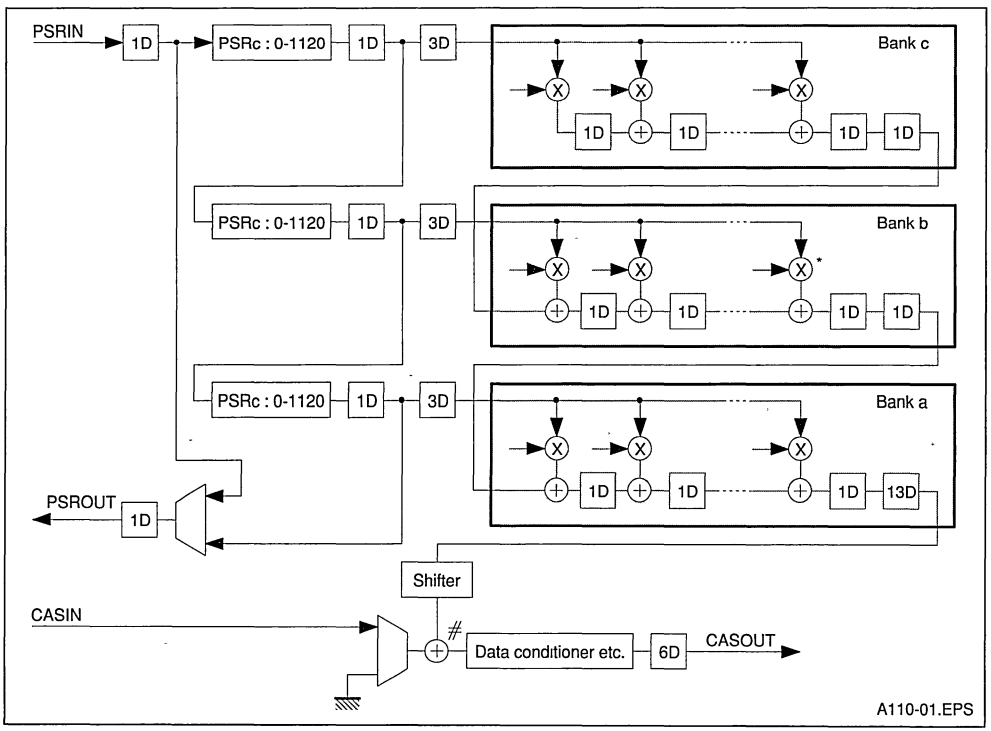
2.2 Two dimensional operation of an IMS A110

For two dimensional operation the first delay (PSRc) is again set to some arbitrary value; however, the setting of PSRa and PSRb is dependant on the line length in pixels of the image being processed. It turns out that in order to achieve a rectangular convolution window the number of delays to be programmed into PSRa and PSRb is equal to the line length in pixels plus the length of the MAC pipelines (seven stages). For example if the screen width of the image to be processed is 512 pixels then the delay to be programmed into shift registers PSRa and PSRb is 519.

N.B. normally when processing an image with an arbitrary setting of PSRc the delay (latency) through the IMS A110 causes the output image to be incorrectly aligned or skewed. This results in an apparent rotation of the output image in the horizontal plane. To correct this problem PSRc may be adjusted to introduce a suitable number of delays to shift the image into the correct position.

Typically image data is fed into an IMS A110 line by line starting at the top left and ending at the bottom right. Given this definition it may be seen that the first MAC stage in each row is processing the data nearest the left hand side of the screen (the oldest data) and that the last MAC stage in each row is processing the data nearest the right hand side of the screen (the newest data). In a similar fashion the first row is always processing the newest data (the data nearest the bottom of the screen) and the last row is always processing the oldest data (the data nearest the top of the screen). It is important to bear in mind these relationships when programming IMS A110s, otherwise the operation being performed on an image may not be what was expected.

Figure 1 : Block Diagram of the IMSA110s



3. FUNDAMENTALS OF CASCADING IMSA110s

Consider a single IMSA110 configured to perform some task on a stream of data values. The filter kernel formed by the coefficients may be thought of as a block passing over the data. To produce bigger filters it is necessary to join a number of separate blocks together. This may be achieved by connecting together a number of IMSA110s, as shown in Figure 2, and configuring them suitably. In order to create a contiguous filter kernel (i.e. a filter without overlap or gaps) it is essential that the route between PSRin and PSRout for each device is programmed correctly and that the internal delay lines are programmed to the correct lengths.

To assist in the calculation of the delays to be programmed into the programmable shift registers it is convenient to define a reference data path through the MAC of any given IMSA110. In this document, unless specified, the reference path is taken to be from the input to the multiplier marked with an asterisk (*) in Figure 1 to the cascade adder marked with a hash (#).

In addition before embarking on any calculation it is necessary to know the following:

1 The delay between PSRin and PSRout when the data is routed directly from PSRin to PSRout without passing through the programmable shift registers. This delay is known as D_D.
2 The delay along the reference path. This delay is known as D_R.
3 The delay through the backend between cascade in and cascade out. This delay is known as D_B.
4 The locations of the other inherent delays within IMSA110s.
5 The meaning of line length, kernel width and kernel height. See Figure 3 for a definition of these terms.

Figure 1 shows a functional block diagram of an IMSA110 with all the inherent delays included. From this diagram it is possible to calculate the value of the three delay constants as shown in table 1.

Table 1

D_D=1+1	D_R=(1+1)+(7+1)+(7+13)	D_B=6
D_D=2	D_R=30	

Figure 2 : Standard Connection for Cascading IMSA110s

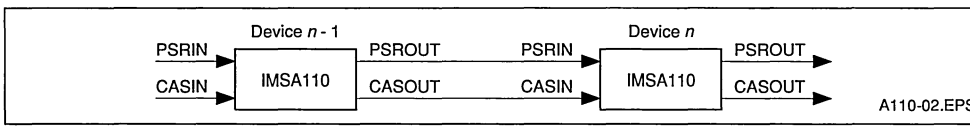
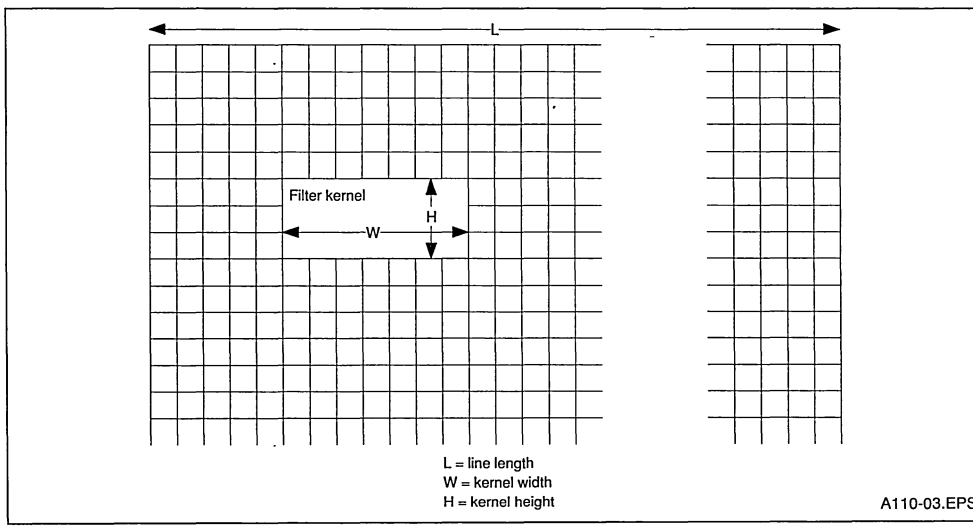


Figure 3 : Depiction of Line Length, Kernel Width and Kernel Height



4. CASCADING IMSA110s TO PRODUCE LONG ONE DIMENSIONAL FILTERS

A single IMSA110 is capable of producing a one dimensional filter with up to 21 taps (shorter filters may be made by setting unrequired coefficients to zero). To create longer filters it is necessary to cascade a number of IMSA110s together. Each additional device added to the cascade gives an additional 21 taps allowing filters of almost unlimited size to be built from simple building blocks. To develop the delays required to be set up in a one dimensional cascade the system shown in Figure 4 will be considered. This system only contains two devices but will be examined in a general way so that rules may be developed for cascades of arbitrary length. It has already been mentioned how to set up the delays to achieve one dimensional convolution in a single device. Fortunately, in cascades of IMSA110s the data relationships within each device are the same as those which would exist inside a single non cascaded device processing the same data. Hence, in the one dimensional cascade under consideration the delays programmed into PSRa and PSRb of each device are zero.

In order to cascade IMSA110s into long one dimensional filters the data is normally routed directly from the input to the output of each device without passing through the programmable shift registers, as shown in Figure 4. It may be seen that each piece of data takes two routes through the cascade. One route generates partial results via the MAC of device n-1 and the other via the MAC of device n. These partial results are eventually combined at the cascade adder in the backend of device n. To produce the correct result it is important that these two separate data streams are aligned correctly.

Assuming that the delay in the PSRc of device n-1 is x_{n-1} and that the delay in PSRc of device n is x_n, it is desired to calculate the relationship between these delays for correct combination of the partial results. Consider an item of data when it reaches device n-1. The delay before the component due to this data, flowing via the reference path in device n-1, reaches the cascade adder of device n is:

$$D_{n-1} = 1+x_{n-1}+1+3+D_R+D_B$$
$$D_{n-1} = 41+x_{n-1}$$

Similarly the delay before the component due to this data, flowing via the reference path in device n, reaches the cascade adder of device n is:

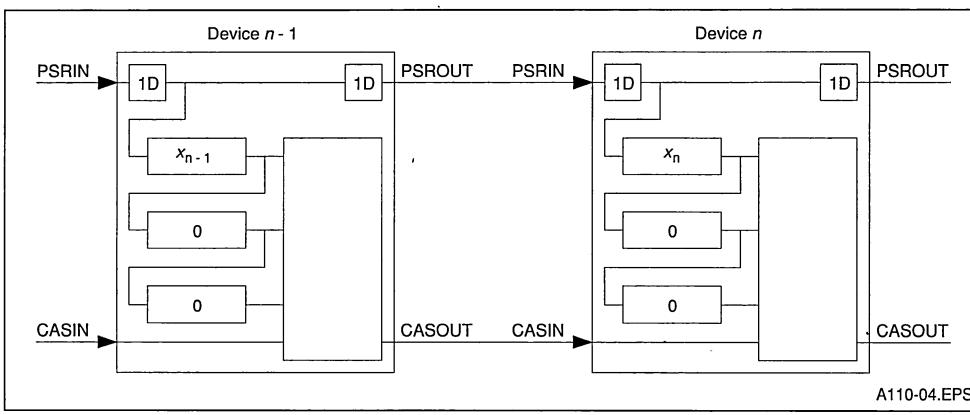
$$D_n = D_D+1+x_n+1+3+D_R$$
$$D_n = 37+x_n$$

Now, for a contiguous convolution kernel, it is desired for the results flowing via the MAC of device n-1 to arrive at the cascade adder of device n, 21 clock cycles behind those which have come from the other route. Hence:

$$D_{n-1}-D_n = 21$$
$$41+x_{n-1}-37-x_n = 21$$
$$x_{n-1} = x_n+17$$

This means that the PSRc of device n-1 must be programmed with the value which is in PSRc of device n plus a fixed constant of 17. This rule may be extended to take into account any number of devices providing that the maximum length of the delay lines is not exceeded. The PSRc of the last device in the cascade may be programmed to an arbitrary value (normally zero) providing the maximum length of the first PSRc delay in the cascade is not exceeded.

Figure 4 : Direct Data Path Connection for Cascading IMSA110s



For example consider the problem of filtering a data stream with a 50 tap filter. This could be achieved by cascading three IMSA110s. Typical delays which would have to be programmed into the devices are given in table 2.

Table 2

	Device 1	Device 1	Device 2
PSRa	0	0	0
PSRb	0	0	0
PSRc	34	17	0

5. CASCADING IMSA110s TO PRODUCE WIDER TWO DIMENSIONAL FILTERS

A single IMSA110 is capable of filtering an image with a two dimensional kernel which has a maximum width of seven cells (narrower filters may be made by setting unrequired coefficients to zero). To create wider filters it is necessary to cascade a number of IMSA110s together. Each additional device added to the cascade increases the maximum width by an additional 7 cells, allowing filters of almost unlimited width to be created.

The connections required to cascade IMSA110s into horizontal cascades may be seen in Figure 4. It may be noted that the connections for this type of cascade are identical to those presented in section 4 for one dimensional cascading. The difference in function is achieved by changing the delays present in the programmable shift registers. It was mentioned in section 2 that for two dimensional filtering using a single device the length of PSRa and PSRb have to be programmed to the line length plus seven. Hence to ensure correct alignment of the rows of the filter in a horizontal cascade it is necessary that PSRa and PSRb of each of the devices must also be set to this value. In order to cascade horizontally the pixel data is normally routed directly from the input to the output of each device without passing through the programmable shift registers. As before it may be seen that each item of data (pixel) takes two routes through the cascade. By assuming that the delay in the PSRc of device n-1 is x_{n-1} and that the delay in PSRc of device n is x_n, then the route delay equations derived are the same as those calculated in section 4.

$$D_{n-1} = 41+x_{n-1}$$
$$D_n = 37+x_n$$

Now, for a contiguous convolution kernel, it is desired for results flowing via the MAC of device n-1 to arrive, at the cascade adder of device n, 7 clock cycles behind those which have come from the other route. This may be achieved by ensuring that the data passing via MAC n-1 takes 7 cycles longer than data passing via the MAC n route. Hence:

$$D_{n-1}-D_n = 7$$
$$41+x_{n-1}-37-x_n = 7$$
$$x_{n-1} = x_n+3$$

This means that the PSRc of device n-1 must be programmed with the value which is in PSRc of device n plus a fixed constant of 3. This rule may be extended to cascade any number of devices providing that the maximum length of the delay lines is not exceeded. The value programmed into the PSRc of the last device in the cascade is arbitrary (normally adjusted to deskew the output image) but must not be set so high that the PSRc of the first device in the cascade exceeds its maximum.

For example consider the problem of filtering a 1024 pixel wide image with a 15×3 filter kernel. This could be achieved by cascading three IMS A110s into a horizontal cascade. Typical delays which would have to be programmed into the devices are given in table 3.

Table 3

	Device 1	Device 2	Device 3
PSRa	1031	1031	1031
PSRb	1031	1031	1031
PSRc	6	3	0

6. CASCADING IMSA110s TO PRODUCE HIGHER TWO DIMENSIONAL FILTERS

The maximum height of a two dimensional filter kernel produced by a single IMSA110 is three cells. This is restricting in some applications but, may be easily overcome by cascading a number of IMSA110s into a single vertical strip. The theoretical maximum height of filter which can be created is equal to three times the number of devices cascaded. Hence the vertical filter size is limited only by the number of devices used.

To develop the delays required to be setup in a vertical cascade the system shown in Figure 5 will be considered. This system only contains two devices but will be examined in a general way so that rules may be developed for cascades of arbitrary length. It was mentioned in section 2 that for two dimensional filtering using a single device the length of PSRa and PSRb have to be programmed to the line length plus seven (L+7). Obviously to ensure correct alignment of the rows of the filter in a vertical cascade it is necessary that PSRa and PSRb of each of the devices must also be set to this value.

To cascade vertically the pixel data is normally routed from the input to the output of each device via the programmable shift registers (see Figure 5). Again it may be seen that each pixel takes two routes through the cascade. One route generates partial results via the MAC of device n-1 and the other via the MAC of device n.

These partial results are eventually combined at the cascade adder in the backend of device n. In order to produce the correct result it is important that these two data streams are aligned correctly.

Assuming that the delay in the PSRc of device n-1 is x_{n-1} and that the delay in PSRc of device n is x_n, it is desired to calculate the the relationship between these delays for correct combination of the partial results. Consider a pixel when it reaches device n-1. The delay before the component due to this pixel, flowing via the reference path in device n-1, reaches the cascade adder of device n is:

$$D_{n-1} = 1+x_{n-1}+1+3+D_R+D_B$$
$$D_{n-1} = 41+x_{n-1}$$

Similarly the delay before the component due to this pixel, flowing via the reference path in device n, reaches the cascade adder of device n is:

$$D_n = 1+(x_{n-1}+1)+(L+7+1)+(L+7+1)+1+1+(x_n+1)+3+D_R$$
$$D_n = 54+2L+x_{n-1}+x_n$$

But for a contiguous convolution kernel it is desired for results flowing via MAC n to arrive, at the cascade adder of device n, three line lengths after those which have come from the other route. This may be achieved by ensuring that the data passing via MAC n takes 3L (where L is the line length in pixels) cycles longer than data passing via the MAC n-1 route. Hence:

$$D_n-D_{n-1}=3L$$
$$54+2L+x_{n-1}+x_n-41-x_{n-1}=3L$$
$$x_n=L-13$$

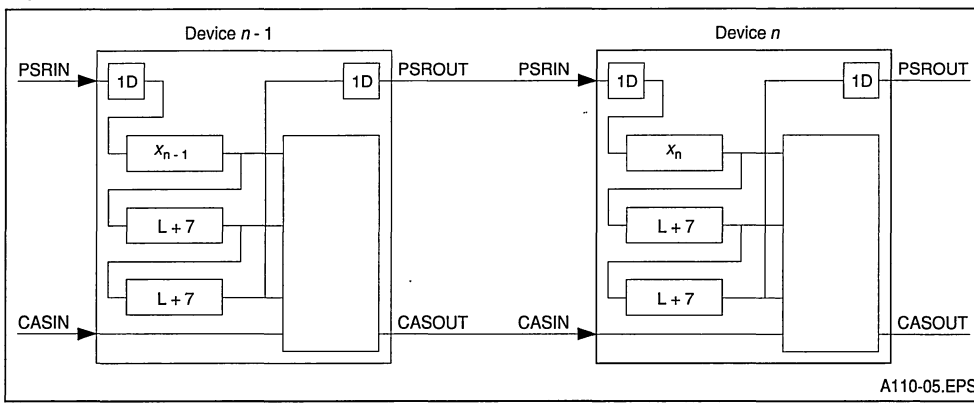
This means that the PSRc of device n must be programmed with a value which is equal to the line length minus a fixed constant of 13. This rule may be extended to cascades containing any number of devices providing that the maximum length of the delay lines is not exceeded. N.B. the setting of the PSRc of the first device in the cascade is arbitrary and may be adjusted to deskew the output image.

For example consider the problem of filtering a 512 pixel wide image with a 7×7 filter kernel. This could be achieved by cascading 3 IMS A110s into a vertical cascade. Typical delays which would have to be programmed into the devices are given in table 4.

Table 4

	Device 1	Device 2	Device 3
PSRa	519	519	519
PSRb	519	519	519
PSRc	0	499	499

Figure 5 : Indirect Data Path Connection for Cascading IMSA110s



7. Cascading IMSA110s to produce wider and higher two dimensional filters

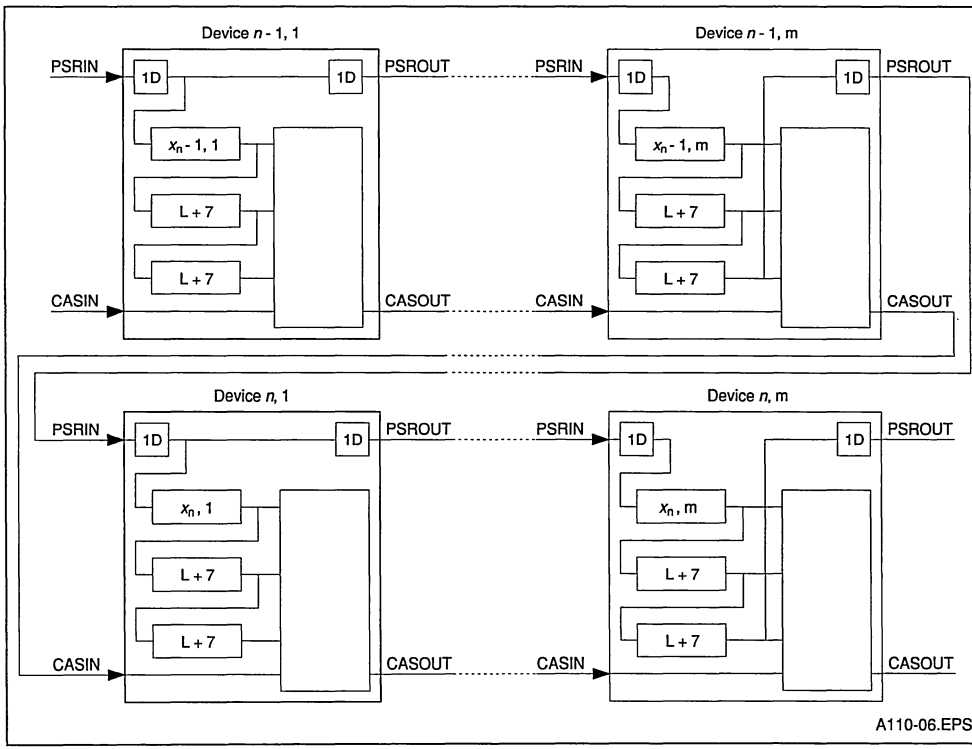
To produce filters which are both wider and higher than allowed by a single IMSA110 it is possible to cascade a number of the wider filters discussed in section 5 into a vertical strip.

The connections required to cascade IMSA110s into two dimensional cascades may be seen in Figure 5. The system shown has arbitrary width but only two rows of devices allowing a maximum filter height of six cells. However the system will be examined in a general way so that rules may be developed for cascades of arbitrary height. It may be noted that across each row, except for the last device, direct connection is used between PSRin and PSRout. The last device uses the indirect route via the programmable shift registers to connect to the first device of the next row. Since each row of this cascade consists of a horizontal cascade the rules developed for the delays in such a cascade (see section 6) apply to each row of this larger configuration. However, the relationship between the delays in the vertical direction requires careful consideration.

Assuming that the array of IMS A110s contains M devices in the horizontal direction and that the delay in PSRc of each device is as shown in Figure 6, it is desired to calculate the relationship between these delays for correct combination of the partial results generated by each row within the cascade. Consider a pixel when it reaches device n-1,1. The delay before the first component due to this pixel, flowing via the reference path in device n-1,1, reaches the cascade adder of device n,1 is:

$$D_{n-1,1} = 1+x_{n-1,1}+1+3+D_R+D_BM$$
$$D_{n-1,1} = 35+6M+x_{n-1,1}$$

Figure 6 : Connections for Cascading IMSA110s into Wider and Higher 2-D Filters



Similarly the delay before the component due to this pixel, flowing via the reference path in device n,1, reaches the cascade adder of device n,1 is:

$$D_{n,1} = 2(M-1) + 1 + (x_{n-1}, M+1) + (L+7+1) + (L+7+1) + 1 + 1 + (x_{n,1}+1) + 3 + D_R$$
$$D_{n,1} = 52+2M+2L+x_{n-1,M}+x_{n,1}$$

But for a contiguous convolution kernel it is desired for results flowing via MAC n,1 to arrive, at the cascade adder of device n,1 a period of 3L clock cycles after those which have come from the other route. Hence:

$$D_{n,1}-D_{n-1,1} = 3L$$
$$52+2M+2L+x_{n-1,M}+x_{n,1}-35-6M-x_{n-1,1} = 3L$$
$$17-L-4M+x_{n-1,M}+x_{n,1} = x_{n-1,1}$$

Now it is also known from section 5 that any given device in a row except the final device has PSRc programmed to 3 more than the device which follows. This leads to the following relationship between the delays programmed into the first and the last devices of the top row:

$$x_{n-1,1} = x_{n-1,M}+3(M-1)$$

By substituting this result into the previous result gives: $x_{n,1}=7M+L-20$

This means that the PSRc of device n,1 must be programmed with the value which is equal to 7 times the number of devices cascaded horizontally plus the line length minus a fixed constant of 20. This rule may be extended to cascades containing any number of devices providing that the maximum length of the delay lines is not exceeded. N.B. the setting of PSRc of the right most device in the first row is arbitrary, but is normally adjusted to deskew the output image.

For example consider the problem of filtering a 512 pixel wide image with a 9×9 filter kernel. This could be achieved by cascading six IMS A110s into a cascade containing three rows of two devices. Typical delays which would have to be programmed into the devices are given in table 5.

Table 5

	Device 1,1	Device 1,2	Device 2,1	Device 2,2	Device 3,1	Device 3,2
PSRa	519	519	519	519	519	519
PSRb	519	519	519	519	519	519
PSRc	3	0	506	503	506	503

8. Cascading IMSA110s to perform multi pass filtering operations

In addition to being able to cascade IMSA110s for increased filter size it is also possible to cascade devices to perform multi pass filtering operations. For example consider the problem of edge detection in a noisy image. This task is often performed in two stages the first is low pass filtering to reduce the amount of noise and the second is the edge detection operation. This complete task may be performed by cascading two IMSA110s as shown in Figure 7. Note that only an eight bit window of CASout from the first device is connected to PSRin of the second device.

To configure such a cascade to perform the double filtering operation each device is considered separately and the delays are setup as described in section 2. For the example under consideration the coefficients of the first device are configured to perform the low pass filter operation while the coefficients of the second device are configured as an edge detector.

This technique of multi pass filtering can obviously be extended to include more devices or it may be combined with the cascading techniques discussed in earlier sections to allow multi pass filtering with larger filter sizes.

It is possible to use a single device for multi pass filtering. This technique works by feeding back alternate cascade outputs to PSRin, and making use of bank swapping. Figure 8 shows the basic setup. The disadvantages of this method are:

1 The maximum data throughput is halved.
2 The maximum filter size is reduced.
3 External logic is required.

To setup such a system requires careful programming to achieve the desired result. For example consider the problem of passing the local averaging filter kernel shown below over an image twice.

1	1	1
1	1	1
1	1	1

It may be shown using similar techniques to those presented earlier that the delays to be programmed into the programmable shift registers a and b are: 2L+7

This value is equal to twice the line length plus the length of the MAC pipelines. N.B. logical reasoning would have lead to the same result by considering that the data rate within the device is equal to twice the rate of the applied image data.

Figure 7 : Cascading IMSA110s for Multi-pass Filtering

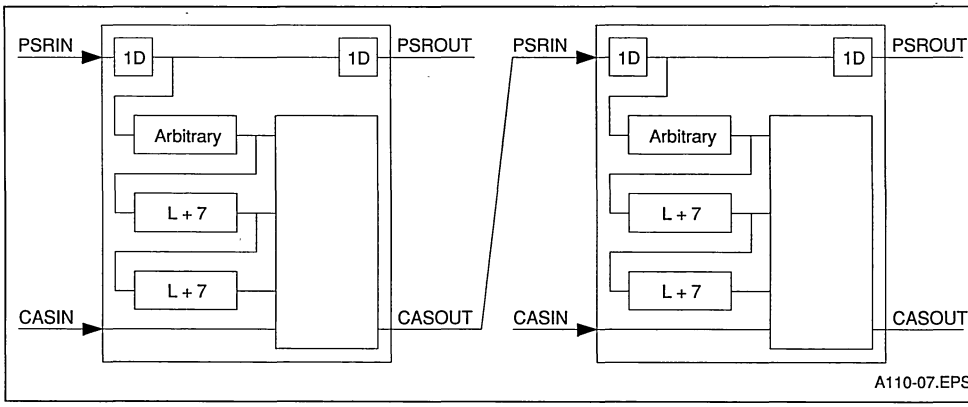
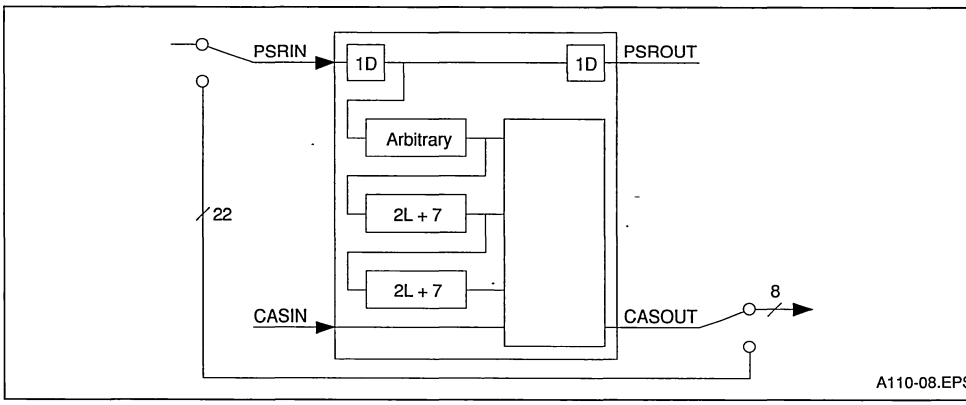


Figure 8 : Multi-pass Filtering by using Feedback



To create the correct filter kernels it is very important that the coefficient registers are programmed correctly. Each filter is programmed into one of the two coefficient banks, and every odd coefficient must be set to zero otherwise the two interleaved data streams will corrupt each other. The table below shows how the coefficients should be programmed for the example under consideration.

CR0	a	1	0	1	0	1	0	0
	b	1	0	1	0	1	0	0
	c	1	0	1	0	1	0	0

CR1	a	1	0	1	0	1	0	0
	b	1	0	1	0	1	0	0
	c	1	0	1	0	1	0	0

9. CASCADING IMS A110s FOR INCREASED DATA PRECISION

In some high precision applications the 8 bit word length of a single IMSA110 is not sufficient. This section presents three techniques to overcome this problem. The first two combine IMSA110s with simple external hardware, the last one requires no external hardware but does place certain restrictions on the coefficients and the data.

9.1 Increasing data precision with an external 22 bit adder

The first technique makes use of an external 22 bit adder in the configuration shown in Figure 9.

At the input each 16 bit input value is split into two

8 bit words one containing the least significant 8 bits and the other containing the most significant 8 bits. Each of these 8 bit data streams is fed into an IMSA110. If the data is unsigned then both of the devices must be set to unsigned data operation. However, if the data is signed then in order to correctly process the data and preserve the sign information it is necessary for the least significant byte to be processed as unsigned data and the most significant byte to be processed as signed data (see Figure 9). This may be easily achieved by setting or clearing bit 2 of the SCR register in each IMSA110 as appropriate. The 22 bit partial results from each device are combined by making use of a 22 bit adder. This adder forms the sum of the top 14 bits (sign extended to 22 bits if signed data is being used) of the least significant partial result and the full 22 bits of the most significant partial result to give the upper 22 bits of the final result. This is combined with the lower 8 bits of the least significant partial result to give the complete 30 bit result. See Figure 10 for a graphical representation of this.

Note that for signed data, the least significant partial result must be signed extended to 30 bits as shown in Figure 9. The sign extension is easily achieved by connecting the most significant bit of the least significant partial result to the most significant 8 bits of the adder input.

This technique may be extended to give data precisions above 16 bits, however, such precisions are rarely used in practice. Sometimes it may be desired to combine a bigger filter size, as discussed in earlier sections, with increased precision. Such a system is simple to create and just involves replacing each IMSA110 in Figure 9 with the appropriate cascade of devices. Similarly multi pass filtering, as discussed in section 8, may be combined with increased precision. This is achieved by selecting a 16 bit window from the output of the system shown in Figure 9 and feeding this into the input of another high precision stage.

9.2 Increasing data precision with an external delay line

As an alternative to using an external adder it is possible to make use of the cascade adder built into each IMSA110 and an external delay line (of length D_B) as shown in Figure 11:

The rules discussed earlier in this section about signed data apply equally to this configuration. This means that if signed data was being processed then the left and right hand devices in the diagram would have to be configured for unsigned and signed operation respectively. Also CASin of device n would have to be sign extended to 22 bits as described in section 9.1. The one other consideration when increasing the data precision in this way is the number delays required in the programmable shift registers of each device.

Figure 9 : Cascade of IMSA110s for Increased Data Precision (signed data)

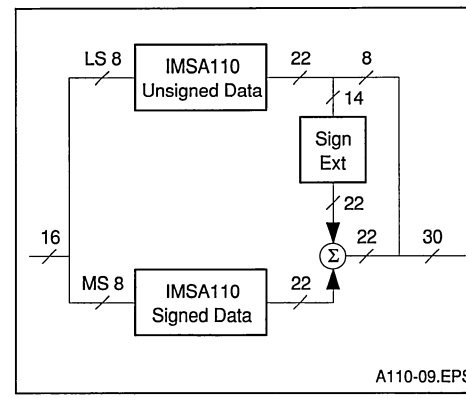


Figure 10 : Calculation of the Final Output

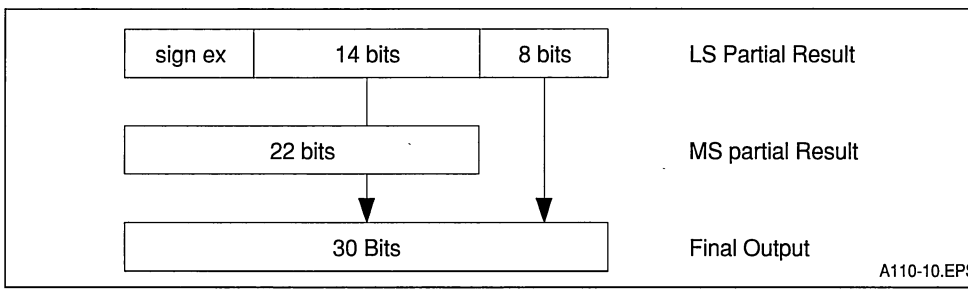
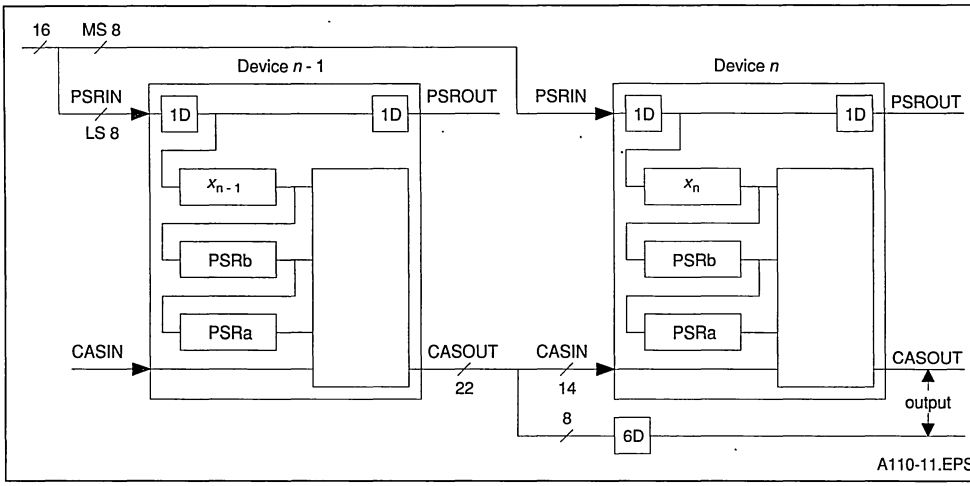


Figure 11 : Alternative Cascade of IMSA110s for Increased Data Precision (unsigned data)



Obviously the setings of PSRa and PSRb are not affected by the presence of another device and are setup as described in section 2. The setting of PSRc for each device however is important, and incorrect setting will result in erroneous calculation of the most significant 22 bits of the result.

Assuming that the delay in the PSRc of device n-1 is x_{n-1} and that the delay in PSRc of device n is x_n, it is desired to calculate the relationship between these delays for correct combination of the partial results. Consider an item of data when it reaches device n-1. The delay before the component due to this data, flowing via the reference path in device n-1, reaches the cascade adder of device n is:

$$D_{n-1} = 1+(x_{n-1}+1)+3+D_R+D_B = D_{n-1} = 41+x_n$$

Similarly the delay before the component due to this data, flowing via the reference path in device n, reaches the cascade adder of device n is:

$$D_n = 1+(x_n+1)+3+D_R = D_n = 35+x_n$$

Now for the data to be correctly aligned at the cascade adder of device n the delay along each path must be the same. Hence:

$$D_{n-1}-D_n = 0$$
$$41+x_{n-1}-35-x_n = 0$$
$$x_n = x_{n-1}+6$$

This means that the PSRc of device n must be programmed with the value which is in PSRc of device n-1 plus a fixed constant of 6.

Obviously this technique of increasing data precision may be extended beyond 16 bits, or may be combined with other cascading techniques to give larger filter sizes etc

9.3 Increasing data precision with no external hardware

If the data and coefficients are such that only 22 bits or less are required to represent the result then it is possible to increase the data precision with no external hardware. The connections required are similar to those shown in Figure 11. However, the 6 stage delay must be removed and the full 22 bits of CASout from the first device must be connected to CASin of the second device. To correctly sum the two contributions of the result, it is necessary to left shift the MAC output of the second device 8 places to the left. This shift is easily performed using the shifter in the second device, however, care must be taken to ensure that overflow does not occur. If such an overflow does occur then it will not be detected.

10. CASCADING IMSA110s FOR INCREASED COEFFICIENT PRECISION

Section 9 described three different techniques for increasing data precision by cascading IMSA110s. In this section three very similar techniques are presented for increasing coefficient precision.

10.1 Increasing coefficient precision with an external 22 bit adder

The first method makes use of an external 22 bit adder as shown in Figure 12. At the input each 8

bit value is fed to PSRin of both the IMS A110s. The device at the top of the diagram is programmed with the least significant 8 bits of the coefficients and the device at the bottom is programmed with the most significant 8 bits of the coefficients. If the coefficients are unsigned then both of the devices must be set to unsigned coefficient operation. However, for signed coefficients, in order to correctly process the data and preserve the sign information it is necessary for unsigned and signed coefficient operation to be set in the top and bottom devices respectively (see Figure 12). This may be easily achieved by setting or clearing bit 3 of the SCR register in each IMS A110 as appropriate. Also, sign extension must be performed as described in section 9.1. The 22 bit partial results are then combined in exactly the same fashion as described in section 9.

As discussed for increased data precision this technique may be extended to more than 16 bits of accuracy if required, or may be adapted to make use of increased filter sizes etc. For very high precision systems increased coefficient and data precision may be combined to give very accurate results.

10.2 Increasing coefficient precision with an external delay line

The second method makes use of a delay line in a very similar configuration to that discussed in the previous section. A diagram showing the setup may be seen in Figure 13.

The rules discussed earlier in this section about signed coefficients still apply in this configuration. Hence if signed coefficients are required then the left and right hand devices in the diagram have to be configured for unsigned and signed coefficient operation respectively and the sign must be extended appropriately. The calculation of the setting of PSRc for each device may be calculated in the same manner as described in the previous section. When the calculation is performed the following relationship is developed:

$$subn = x_{n-1}+4$$

Figure 12 : Cascade of IMSA110s for Increased Coefficient Precision (signed coefficient)

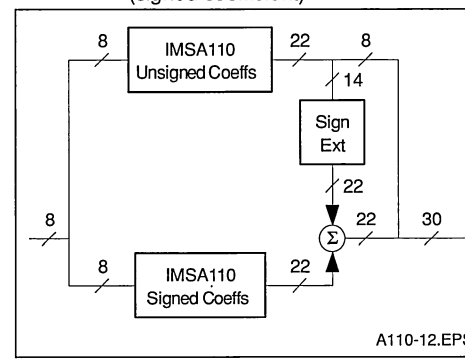
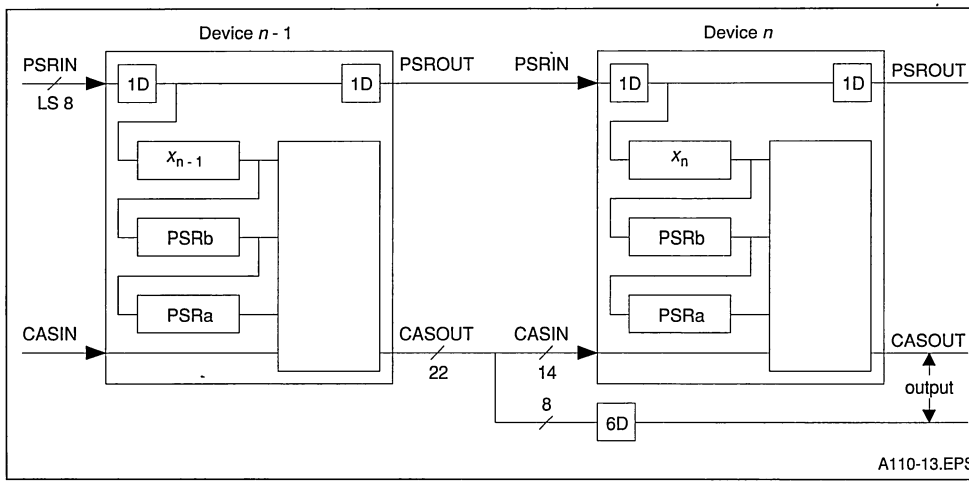


Figure 13 : Alternative Cascade of IMSA110s for Increased Coefficient Precision (unsigned coefficients)



This means that the PSRc of device n must be programmed with the value which is in PSRc of device n-1 plus a fixed constant of 4.

Obviously this technique may be extended for more precision or adapted using information presented in earlier sections to give increased filter size, multi pass filtering etc.

10.3 Increasing coefficient precision with no external hardware

It was discussed in section 9 how to achieve increased data precision without using any external hardware. Since exactly the same technique may be applied to give increased coefficient precision duplicate details are not given here.

11. SUMMARY

This document has attempted to describe some of the many ways in which IMSA110s may be cascaded to yield even higher performance. Obviously it has not been possible to discuss every possible configuration but hopefully the examples discussed should have provided both an insight into the extensive capabilities of these devices when cascaded, and some simple rules to allow easy setting up of some of the most common forms of cascades.

THE IMSA110 BACK-END POST PROCESSOR

SUMMARY

I. INTRODUCTION

The IMSA110 consists of a high performance configurable array of multiply-accumulators (420 MOPs), three programmable length 1120 stage shift registers and a versatile backend post processing unit. All these features are controlled from a microprocessor interface. The comprehensive on-chip facilities ensure that a single device is capable of dealing with many tasks commonly found in the fields of signal and image processing.

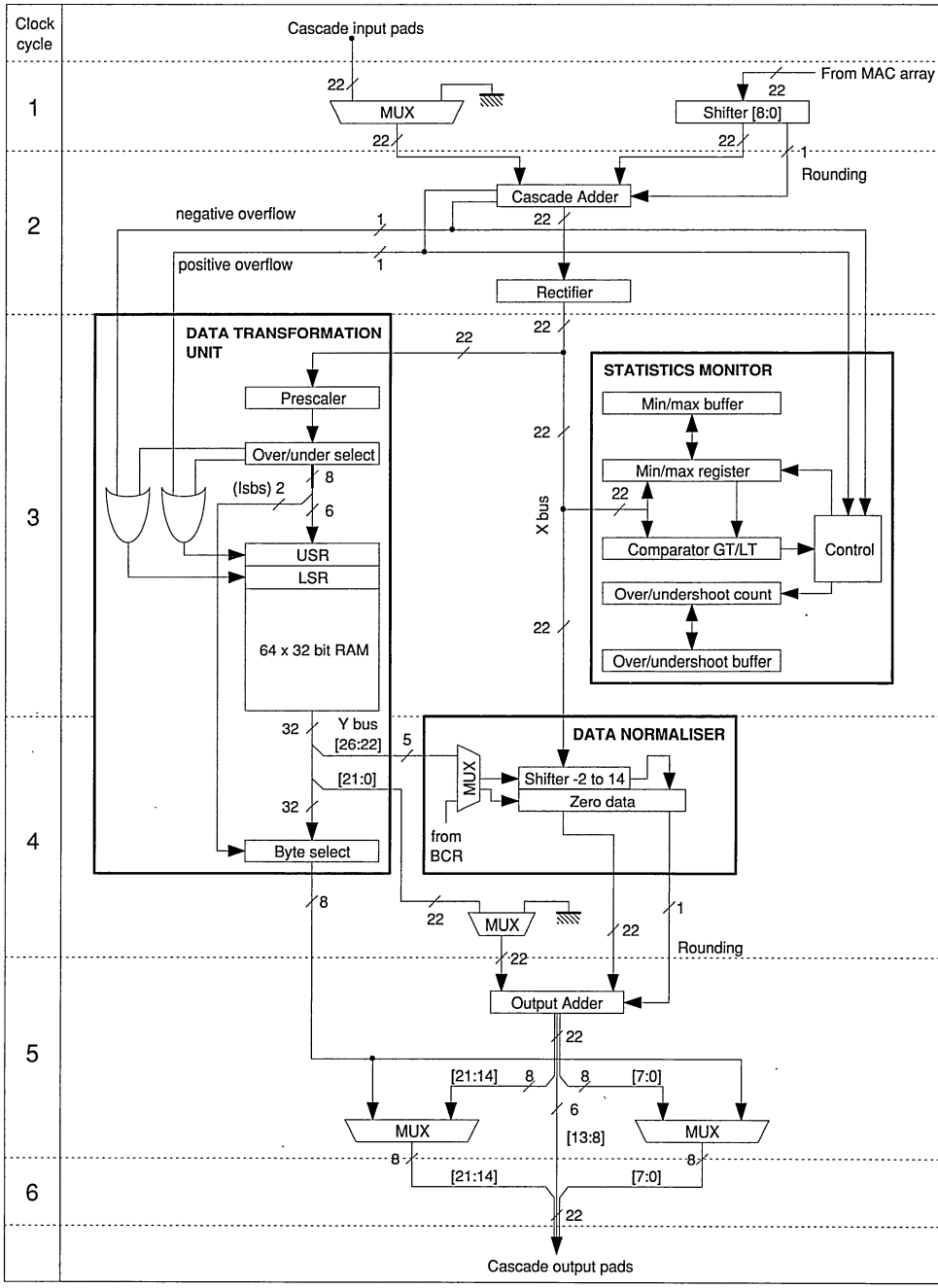
The backend post processing unit gives the IMSA110 a high degree of flexiblility, especially for image processing applications. This document describes by example some of the uses of the backend post processor.

Unless specified otherwise all the examples considered will be based around image processing applications with 8 bits per pixel being used to represent the image data.

2. DESCRIPTION OF THE BACK-END POST PROCESSOR

Figure 1 shows the functional blocks and interconnections which are present within the backend post processor of the IMSA110.

Figure 1 : Detailed Block Diagram of the Back-end Post Processing Unit



This diagram can be broken down into 4 main sections, the input block, statistics monitior, data conditioning unit and output block. A brief description of each of these major sections is given below, for full details reference should be made to the data sheet.

2.1 Input Block (shifter, cascade adder and rectifier)

Data from the MAC array encounters the shifter when it enters the input block. The shifter is capable of up to 8 arithmetic shifts in either direction. When shifting left it is possible for an overflow to occur. Such an overflow is not detected by the device, hence it is left to the user to ensure that unintentional overflows do not occur. When shifting right rounding is applied to improve the accuracy of the device. The magnitude and direction of the shift are controlled by BCR0[5..1] as described in the data sheet.

The output data from the shifter is fed into the cascade adder. Here it is added to both the rounding bit generated by the shifter and the data applied to either the cascade input bus or zero depending on the setting of BCR0[0]. Should the result of the 22 bit signed addition be greater than 2^{21}-1 then a positive overflow is generated. Similarly if the result is less than -2^{22} a negative overflow is generated.

The output from the cascade adder can be optionally full or half wave rectified depending on the setting of BCR0[7..6]. The output of the rectifier drives the X bus. Note that when full wave rectification is being used and the output of the cascade adder is -2^{21} then the output from the rectifier remains as -2^{21}.

2.2 Statistics Monitor

The statistics monitior allows the X bus to be monitored for certain conditions. Four different modes of operation are possible and these are tabulated below:

Mode	BCR1[1]	BCR1[0]
Max Register	0	1
Min Register	0	0
Overshoot Counter	1	1
Undershoot Counter	1	0

When configured to be in max register mode and the X bus exceeds the current threshold in the MMR (max/min register), then the MMR is loaded with the value on the X bus and the counter (OUC) is incremented. If the threshold is not exceeded then no action is taken. Thus assuming the MMR was initially set to -2^{21} its value at some later time is the maximum value which has appeared on the X bus in that period, and the OUC has been incremented by the number of times the threshold has been updated.

If configured to be in min register mode the threshold is updated and the counter incremented whenever the X bus is less than the current threshold. Note that when operating in max/min register mode if a positive or negative overflow occurs then the threshold is not updated since this could leave a misleading value in the MMR.

As an overshoot counter the statistics monitor operates by incrementing the OUC every time the value on the X bus exceeds the threshold in the MMR or if a positive overflow occurs. The OUC is unsigned and will not wrap around, thus behaving as a saturating counter. Similarly when configured to be in undershoot counter mode the OUC is incremented every time the value on the X bus is less than the current threshold.

When overflows occur this is recorded in bits 22 and 23 of the MMR. Positive overflows cause bit 22 to be set while negative overflows cause bit 23 to be set. These bits may be cleared by writing to the MMB copy location.

Direct access to the MMR and OUC via the microprocessor interface is not possible. Instead the reading and writing of these registers is performed by making use of the MMB, CMM, OUB and COU registers. Full details may be found in the data sheet.

2.3 Data Conditioning Unit (data transformation unit and data normaliser)

2.3.1. DATA TRANSFORMATION UNIT

The data transformation unit contains a prescaler, an under/over select detector, a look up table and a byte selector. It may be used on its own to provide arbitrary data mappings of an 8 bit segment of the X bus, or in conjunction with the data normaliser to implement sophisticated dynamic range compression functions.

The prescaler allows an 8 bit field to be selected from anywhere within the 22 bits of the X bus. This 8 bit field is used as an address to the LUT. The way in which the address is treated is defined in SCR[6]. If this bit is set to zero then the address is signed and runs from -128 to 127. Alternatively if this bit is set to one then the address is unsigned and runs from 0 to 255. The over/under select detector monitors the operaton of the prescaler to

ensure that all the significant bits and the sign of the X bus are included within the 8 bit field. If this is not the case then an overselect or underselect signal is generated depending on whether the X bus is positive or negative respectively.
The LUT consists of sixty four 32 bit words. In addition there are a further two 32 bit locations known as the upper and lower saturation registers (USR, LSR). The most significant 6 bits of the address field are used to select one of the 32 bit registers in the LUT. This 32 bit output is known as the Y bus. The least significant 2 bits of the address field are then used to control a byte select on the output. Thus the LUT may be used to provide arbitrary 8bit - 8bit data transformations.
Positive overflows on the X bus or overselects in the prescaler cause the LUT to access the USR overriding the address supplied by the prescaler. Similarly negative overflows and underselects cause the LUT to access the LSR. When such conditions occur the byte select control is also overridden thus causing the most significant byte (byte 3) of the appropriate saturation register to appear on the byte wide output of the data transformation unit.
The LUT is programmed via the memory interface. The addressing for the LUT corresponds directly to the 8 bit field, assuming that the byte selector is being used. To enable access to the LUT, USR and LSR from the microprocessor interface the LUT access control bit ACR[1] must be set to zero. This forces the Y bus to zero and causes the normaliser to be controlled by BCR3[7..3] regardless of the setting of the dynamic normalisation bit. Once the LUT has been programmed the LUT access control bit may be reset to one thus allowing the LUT to be used in the data transformation unit.

2.3.2 DATA NORMALISER

The data normaliser contains a shifter followed by a zero data unit. The shifter is capable of right shifts of up to 14 bits and left shifts of up to 2 bits. Any amount of shift outside this range invokes the zero data unit which zeros the output of the data normaliser. The amount of shift is specified by one of two 5 bit sources. These are either BCR3[7..3] or bits 26 to 22 of the Y bus. The source currently selected is determined by the setting of BCR3[2].

2.4 Output Unit (output adder and output multiplexers)

2.4.1 OUTPUT ADDER

The output adder takes one of its inputs from the data normaliser (including the rounding bit). The other input is either the least significant 22 bits of the Y bus or zero depending on the setting of BCR3[1]

2.4.2 OUTPUT MULTIPLEXERS

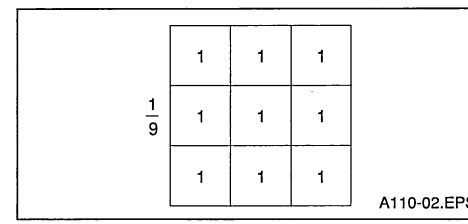
The output multiplexers allow the selected byte from the LUT to be optionally selected to drive either the most or least significant 8 bits of the cascade output pins. This feature is controlled by the setting of BCR2[5..6]. Any cascade output pins not being driven by the selected byte are driven by the appropriate bits of the output adder.

3. USING THE BACKEND OF THE IMS A110s

3.1 Local area averaging

Local averaging is the one of the simplest image filtering operations. A typical local averaging filter may be seen in Figure 2. Although this filter looks very simple to implement on IMSA110s there is one slight problem and that is how to achieve the divide by nine operation. The operation is necessary to ensure that the output image data requires the same number of bits to represent it as the input data.

Figure 2 : Local Averaging Filter Kernel



The IMSA110 is capable of dividing by integer powers of two. Using this capability the $\frac{1}{9}$ could be replaced with $\frac{1}{16}$. Although this would adequately restrict the magnitude of the output data a significant loss of dynamic range could occur. A better solution is to generate an approximation to $\frac{1}{9}$ in the form shown below. Where x represents the coefficient and y the number of righ shift below :

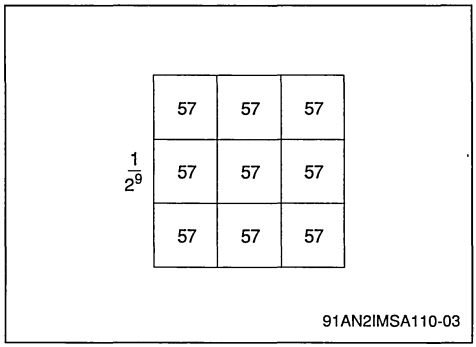
$$\frac{x}{2^y} \approx \frac{1}{9}$$

It may be simply shown that the closest approximation which may be used with IMS A110s is:

$$x = 57$$
$$y = 9$$

By using these values the local averaging kernel to be programmed into the IMSA110 is as shown below:

Figure 3 : Modified Local Averaging Filter Kernel



The division by 2^9 can't be performed by the shifter in the input block since it is only capable of right shifting up to 8 places. The shifter in the normaliser however is capable of right shifting the required nine places.

To configure an IMSA110 so that it performs the local averaging operation used in the above example the following values would have to be programmed into the coefficient and control registers:

Coeff Register	0	1	2	3	4	5	6
CR0a	57	57	57	0	0	0	0
CR0b	57	57	57	0	0	0	0
CR0c	57	57	57	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	x	x	1	1	1	x	0
ACR	0	0	0	0	0	0	x	0
BCR0	x	x	0	0	0	0	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	x	x	x	x	x
BCR3	0	1	0	0	1	0	0	0

x : indicates don't care.

Exactly the same technique may be applied to other filter kernels which require an awkward division. For example the edge enhancement operation shown in Figure 4 requires a division by 5 operation. A modified version of the kernel which may be easily implemented is shown below.

Figure 4 : Edge Enhancement Filter Kernel

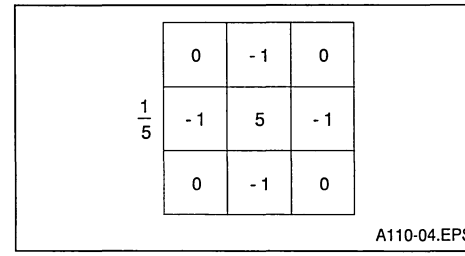
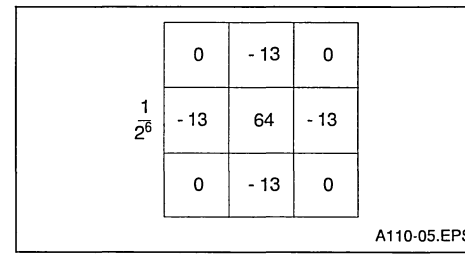


Figure 5 : Modified Edge Enhancement Filter Kernel



3.2 Histogram equalization

Histogram equalization is one example of the wider field of histogram modification [1]. All such operations manipulate the grey levels within an image to generate a new image with a modified grey level histogram. The histogram equalization technique attempts to manipulate the grey levels within an image so that an even spread is obtained across the entire range of intensities. Details of the technique are widely available in the technical press [1] so an in depth discussion will not be provided here.

There are two distinct stages in performing a histogram equalization the second of which IMSA110s are capable of performing. The first stage is the calculation of the transfer function which maps the original image onto the histogram equalized image. The main computational cost involved in this stage is the determination of the original histogram. The second stage requires the implementation of the transfer function to map the grey levels in the input image to the equalized grey levels in the output image.

The transfer function is implemented by making use of the arbitrary 8bit-8bit mapping ability of the LUT present within the IMSA110. The offset of each location in the LUT may be regarded as one of the original grey levels and the value programmed into

that location is the transformed grey level after equalization.
For example suppose that it was desired to use an IMSA110 to perform a histogram equalization on 8 -bit image data applied to the cascade input port with the MAC coefficients programmed to zero. The table below shows the values which would have to be programmed into the main control registers. The output data would appear on the lower 8 bits of the cascade output port.

Register	Data msb .. lsb							
SCR	0	1	x	1	x	x	x	x
ACR	0	0	0	0	0	0	A	x
BCR0	x	x	x	x	x	x	x	0
BCR1	0	0	0	0	0	0	x	x
BCR2	0	1	0	0	0	0	0	0
BCR3	1	0	0	0	0	0	0	0
LUT n	D	D	D	D	D	D	D	D

x : Indicates don't care.
A : Set to 0 to program LUT, set to 1 to allow IMSA110 LUT access.
D : Program with the mapping n ⇒ D[7..0].

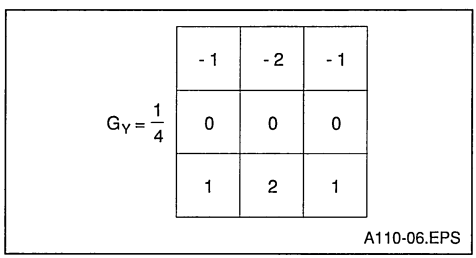
By modifying the transfer function programmed into the LUT many other operations are possible including thresholding and image contouring which are described in sections 3.3 and 3.7 respectively.

3.3 Edge detection and enhancement

3.3.1 EDGE DETECTION

Edge detection is a very important image processing operation since it is often the first stage in feature recognition. For example consider the horizontal edge detector shown in Figure 6. This filter is actually the y component of the Söbel operator. The output (H(x,y)) from the filter when convolved with an image is a measure of the change of intensity in the y direction at each point.

Figure 6 : Y Component of the Söbel Operator



The output at any given point may be positive or negative depending on the direction of the intensity gradient vector at that location. Often when using such a filter to detect vertical edges only the magnitude of the gradient vector is of interest (i.e. its direction is irrelevant). The results may be modified to simply indicate the magnitude by processing the output as shown below.

$$F[x,y]=|H(x,y)|$$

The modulus operation is an ideal example of the use of full wave rectification. The tables below show the configuration of the coefficient and control registers necessary to calculate |H(x,y)|.

Coeff Register	0	1	2	3	4	5	6
CR0a	-1	-2	-1	0	0	0	0
CR0b	0	0	0	0	0	0	0
CR0c	1	2	1	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	x	x	1	0	1	x	0
ACR	0	0	0	0	0	0	x	0
BCR0	1	0	0	0	0	1	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	x	x	x	x	x
BCR3	0	0	0	0	0	0	0	0

x : Indicates don't care.

Typically once an edge detection operator has been convolved with an image it is necessary to make some sort of decision based on the magnitude as to whether an edge exists at each point of the output. The method usually used is known as thresholding [1].
The threshold operation involves mapping all points with a grey level greater than a given threshold to one value (typically 255), and all other points to another value (typically 0). The lookup table as described in section 3.2 provides the ability to perform just such an arbitrary mapping. By modifying the control registers presented above it is possible to do not only the edge detection operation and the full wave rectification, but also to apply an arbitrary threshold all within a single device. The updated table of control registers is shown below:

Registers	Data msb .. lsb							
SCR	0	1	x	1	0	1	x	0
ACR	0	0	0	0	0	0	A	0
BCR0	1	0	0	0	0	1	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	1	0	0	0	0	0	0
BCR3	1	0	0	0	0	0	0	0
LUT n	D	D	D	D	D	D	D	D

x : Indicates don't care.
A : Set to 0 to program LUT, set to 1 to allow IMS A110 LUT access.
D : Set to 0 for n less than or equal to the threshold, set to 1 otherwise.

3.3.2 EDGE ENHANCEMENT

Edge enhancement is often applied to images to either counteract blurring or to produce a sharper looking image which is sometimes aesthetically more pleasing. One filter kernel which gives an edge enhancement may be seen in Figure 5. When this filter is convolved with an image it is possible to generate not only valid positive image data but also negative values under some circumstances. One solution would be to apply full wave rectification to the result however it is generally more acceptable if half wave rectification is applied.

To implement such a filter on an IMSA110 the coefficient and control registers would have to be set up as shown in the following tables.

Coeff Register	0	1	2	3	4	5	6
CR0a	0	-13	0	0	0	0	0
CR0b	-13	64	-13	0	0	0	0
CR0c	0	-13	0	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	x	x	1	0	1	x	0
ACR	0	0	0	0	0	0	0	0
BCR0	0	1	0	0	1	1	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	x	x	x	x	x
BCR3	0	0	0	0	0	0	0	0

x : Indicates don't care.

3.4 Feature recognition

By using the statistics monitor it is possible to get the IMSA110 to see if a given pattern was present within an image. To enable this process to take place a number of things have to be done:

- The MAC coefficients must be configured as a pattern detector for the pattern which is being searched for. If the pattern is large a number of devices can be cascaded [2] to achieve the required window size.
- The statistics monitor must be configured so that it is in max register mode.
- The MMR must be programmed with -2^{21} at the start of the search period (typically at the start of a frame).

As one or more images are processed the MMR register is continually updated to indicate the highest MAC output which has occured so far. When the pattern detector encounters the pattern that it is designed to search for the MAC output should generate a very large output which exceeds a given threshold. This output will be recorded in the MMR. By examining the MMR at the end of the search period (typically at the end of the frame) it is possible to see if the threshold has been exceeded. If this is the case then it is possible to say that the pattern probably occurred somewhere within the data that was processed. The setting of the threshold to achieve reliable operation requires system teaching using known sets of data.

In a similar fashion it is possible to perform feature recognition with the statistics monitor configured as an overshoot counter. In this mode of operation the detection of the desired pattern is indicated by an increase in the value of the OUC (care must be taken to ensure that it does not saturate). The method of setting the threshold at which the overshoot counter is incremented is identical to the description given in the previous paragraph. At first sight it may appear that this method enables the number of occurences of a given pattern to be counted. Unfortunately this is unlikely to be the case for the following reason.

When the pattern being searched for is encountered it is possible for the OUC to be incremented more than once. This is caused by a combination of uncertainty about the pattern and the properties of pattern detectors as decribed below:

- In a typical pattern matching application the pattern is rarely perfect. Degradations from the ideal may be caused by additive noise, distortion of the object, changing lighting conditions etc. To take this into account the threshold is normally set to a value which is low enough to increment the OUC for all likely occurences of the pattern.
- Due to the nature of pattern detectors a large output is not only generated when the detector is coincident with the pattern but quite large outputs can also be generated when it is just off centre.

The combination of these two problems means that each occurence of the pattern could increment the OUC one or more times thus damaging any indication the change in OUC could give about the number of occurences of a pattern.

3.5 Changing conditions compensation

The front end of many automated image processing systems will experience slowly changing input conditions. These may occur due to changing light levels, drifting component tolerances etc. The inclusion of the max/min register modes of the statistics monitor allows the system to automatically compensate for these changes. For example consider a system which uses daylight to illuminate the

field of view. As the day proceeds the output from the camera will change. By spending periods of time monitoring both the maximum and minimum levels in the data stream it is possible to adapt the system to take these changes into account.

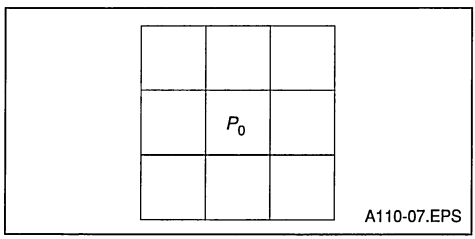
3.6 Binary image procesing

A binary image is one which contains only two grey levels. Typically a binary image is the result of a thresholding operation as described in section 3.3. By making use of the MAC and the backend it is possible to implement a wide variety of different operations some of which are summarised below:

- Isolated pixel removal — removal of all pixels which have no identical neighbour.
- Line linking — bridging of small gaps between pixels.
- Encoding according to connectivity — coding of pixels depending on their connectivity with respect to surrounding pixels.
- Binary thinning including staircase elimination — [3] [4] [5] [6] [7]
- Feature growth — opposite of the above.
- Conway's game of life — the oldest computer game known to man.

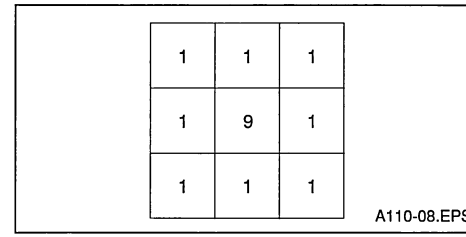
As an example of the techniques involved isolated pixel removal will be examined in more detail. Consider a pixel with its 8 surrounding neighbours as shown in Figure 7. It is assumed that active and inactive pixels are represented by 1 and 0 respectively.

Figure 7 : A Pixel and its 8 closed neighbours



If the central pixel is in the opposite state to all its surrounding neighbours then the value of the central pixel must be toggled. In order to perform the transformation it is necessary to develop a filter kernel which will give a unique output for each of these two condition. One such kernel is shown in Figure 8 below:

Figure 8 : Filter Kernel for Isolated Pixel Removal



By programming the MAC with this kernel the outputs generated when the binary image is applied will range from 0 to 17 inclusive. The two particular cases of special interest are 8 and 9 which correspond to a 0 surrounded by 1s and a 1 surrounded by 0s respectively.

To convert from the output of the MAC to a binary image in the original format use may be made of the LUT. The complete mapping for the LUT and the setting of the main control registers for this example are tabulated below:

Coeff Register	0	1	2	3	4	5	6
CR0a	1	1	1	0	0	0	0
CR0b	1	9	1	0	0	0	0
CR0c	1	1	1	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	1	x	1	1	1	x	0
ACR	0	0	0	0	0	0	A	0
BCR0	x	x	0	0	0	0	0	1
BCR1	0	0	0	0	0	0	x	x
BCR2	0	1	0	0	0	0	0	0
BCR3	1	0	0	0	0	0	0	0
LUT 0-7	0	0	0	0	0	0	0	0
LUT 8	0	0	0	0	0	0	0	1
LUT 9	0	0	0	0	0	0	0	0
LUT 10-17	0	0	0	0	0	0	0	1

x : Indicates don't care.
A : Set to 0 to program LUT, set to 1 to allow IMS A110 LUT access.

3.7 Multilevel thresholding - image contouring

Often it is desired to highlight a number of areas within a single image. Providing that each of the areas occupies a different region of the grey scale then this can be achieved by multi level thresholding (sometimes known as image contouring). Typically such a technique is often used in medical

work. For example consider an X-Ray taken of a patient which may well contain three very distinct regions:

- Clear regions: representing bone.
- Intermediate regions: representing major body organs.
- Dark regions: representing regions where the X-Rays met little resistance.

By using the LUT to provide arbitrary 8bit-8bit data mappings as descibed in sections 3.2 and 3.3 it is possible to assign each of these three regions a separate value. As a further enhancement external hardware could be used to colour each of the three regions. Such colouring can greatly simplify the comprehension of some types of image.

3.8 Dynamic range compression

Consider image data which requires 12 bits to represent each pixel. If it is desired to display such an image on a system which uses only 8 bits per pixel then some form of range compression is required. One solution is to discard the lower 4 bits of each pixel. This would leave the 8 most significant bits for display. If however, the image was dark the lower 4 bits would contain a large proportion of the image data. To throw away the lower 4 bits in such a situation would almost certainly be unacceptable. A better solution in this case would be to use the nonlinear tranformation shown in Figure 9. Using this transformation values between 0 and 63 are unchanged; values between 64 and 1023 are mapped into the range 64 to 183 and values between 256 and 4095 are mapped into the range 184 to 232.

The IMSA110 is capable of performing just such a nonlinear transformation by making use of both the data transformation unit and the data normaliser. The mode of operation which is required is known as dynamic normalisation, this is selected by setting BCR3[2] (enable dynamic normalisation). In this mode the prescaler selects a 6-bit field anywhere within the X bus. This is used as an address to the LUT. Bits 22 to 26 of the output of the LUT are used to control the normaliser block so that the input to the normaliser is dynamically scaled. The output of the normaliser is then added, in the output adder, to the least significant 22 bits of the output of the LUT.

The operation can be viewed as:

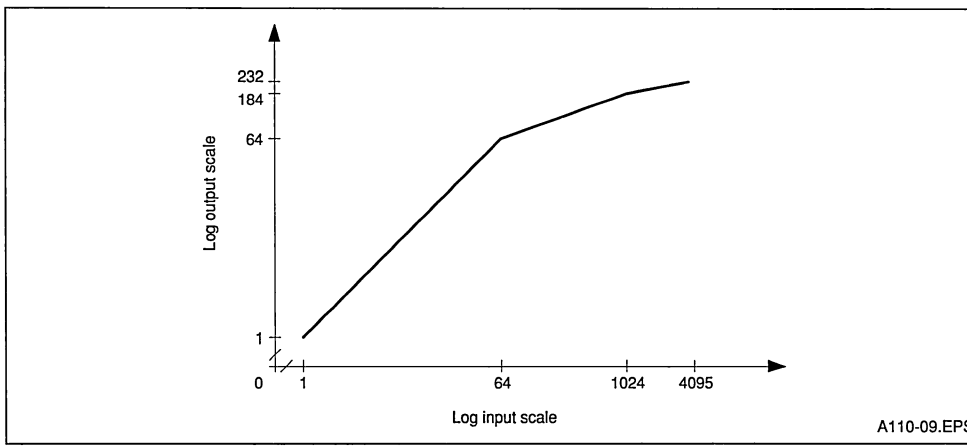
output=(input×scale)+offset

where the scale is provided by bits 22 to 26 and the offset is provided by bits 0 to 21 of the LUT.

To define the transformation function shown in Figure 9 it is necessary to carefully calculate the values to be placed in the LUT. The first stage in this calculation is deciding which slice of the X bus the prescaler is going to select. In this example it will be set so that bits 4 through to 11 are selected. This means that bits 6 to 11 are used as the address for the lookup table. Bearing this in mind it may be seen that in the first segment of the transfer function the LUT address is zero. Since in this segment the scale is 1 (0 right shifts) and the offset is 0 the following four bytes of data must be programmed into the first 32 bit location of the LUT.

	BYTE 3	BYTE 2	BYTE 1	BYTE 0
LUT 0	00	00	00	00

Figure 9 : Typical Dynamic Range Compression Function



The second segment of the transfer function occurs between LUT addresses 1 to 15. In this segment the gradient is 1/8 (3 Right shifts). To ensure that the first and second segment line up correctly it is important to set the offset of the second segment to the correct value.
It may be easily shown that in this case the offset is 56. Thus the data to be programmed into the 15 LUT locations from addresses 1 to 15 is:

	BYTE 3	BYTE 2	BYTE 1	BYTE 0
LUT 1	00	C0	00	38
LUT n	00	C0	00	38
LUT 15	00	C0	00	38

In exactly the same manner the LUT data for the third and final segment of the transfer function may be shown to be:

	BYTE 3	BYTE 2	BYTE 1	BYTE 0
LUT 16	01	80	00	A8
LUT n	01	80	00	A8
LUT 63	01	80	00	A8

The settings of the other main control registers to perform the example transform on data applied to the cascade input port are:

Coeff Register	0	1	2	3	4	5	6
CR0a	0	0	0	0	0	0	0
CR0b	0	0	0	0	0	0	0
CR0c	0	0	0	0	0	0	0

Registers	Data msb .. lsb							
SCR	0	1	x	1	x	x	x	0
ACR	0	0	0	0	0	0	A	0
BCR0	x	x	x	x	x	x	x	0
BCR1	0	0	0	0	0	0	x	x
BCR2	0	0	0	0	0	1	0	0
BCR3	x	x	x	x	x	1	1	0

x : indicates don't care.
A : Set to 0 to program the LUT, set to 1 to allow IMS A110 LUT access.

4. SUMMARY

This document has attempted to describe by example some of the many ways in which the backend post processor of the IMSA110 may be used. It has only been possible to scratch the surface of a handful of applications but hopefully the examples discussed should have provided an insight into both the flexibility and capability of this section of the device.

5. REFERENCES

[1] R. C. Gonzalez, P.Wintz — Digital Image Processing, Addison Wesley.

[2] R. Whitton — Cascading IMS A110s, INMOS.

[3] R. Stefanelli, A. Rosenfeld — Some Parallel Thinning Algorithms For Digital Pictures. Comm ACM 18, 2.

[4] H.E. Lu, P.S.P. Wang — A Comment On Fast Parallel Algorithms For Thinning Digital Patterns. Comm ACM 29, 3.

[5] C.M. Holt, A. Stewart, M. Clint, R.H. Perrott — An Improved Parallel Thinning Algorithm. Comm ACM 30, 2.

[6] R.W. Hall — Fast Parallel Thinning Algorithms: Parallel Speed And Connectivity Preservation. Comm ACM 32, 1.

[7] Z. Guo, R.W. Hall — Parallel Thinning With Two Subiteration Algorithms. Comm ACM 32, 3.

THINNING DIGITAL PATTERNS USING THE IMSA110

SUMMARY

1. INTRODUCTION

1.1 Thinning

Thinning is a very important preprocessing stage of pattern recognition. It is a technique which extracts the basic shapes from images. These shapes are known as skeletons. It attempts to remove all redundant points while maintaining the basic structure and connectivity of the original patterns.

In [1] an algorithm is proposed and modifications to it are described in [2]. The final form of the algorithm has the advantage of being both fast, and suitable for parallel operation.

1.2 The IMSA110

The IMSA110 [3] is a single-chip reconfigurable and cascadable subsystem suitable for many high speed image and signal processing applications. The IMSA110 consists of a configurable array of multiply-accumulators, three programmable 1120 stage shift registers, a versatile post-processing unit and a microprocessor interface for configuration and control purposes.

Figure 1 shows the main processing core of the device. It consists of 21 multiply accumulate stages arranged in three banks of seven. These may be configured as either a 21 stage pipeline or a 3x7 two-dimensional window. The output from the MACs is fed into the backend processing unit. It is this section which allows various data transformations to take place adding greatly to the flexibility of the overall device. The maximum data rate which may be applied to the inputs is 20MHz.

Figure 2 shows a functional block diagram of the backend post processing unit. Complete details may be found in [3].

Figure 1 : IMSA110 Users Model

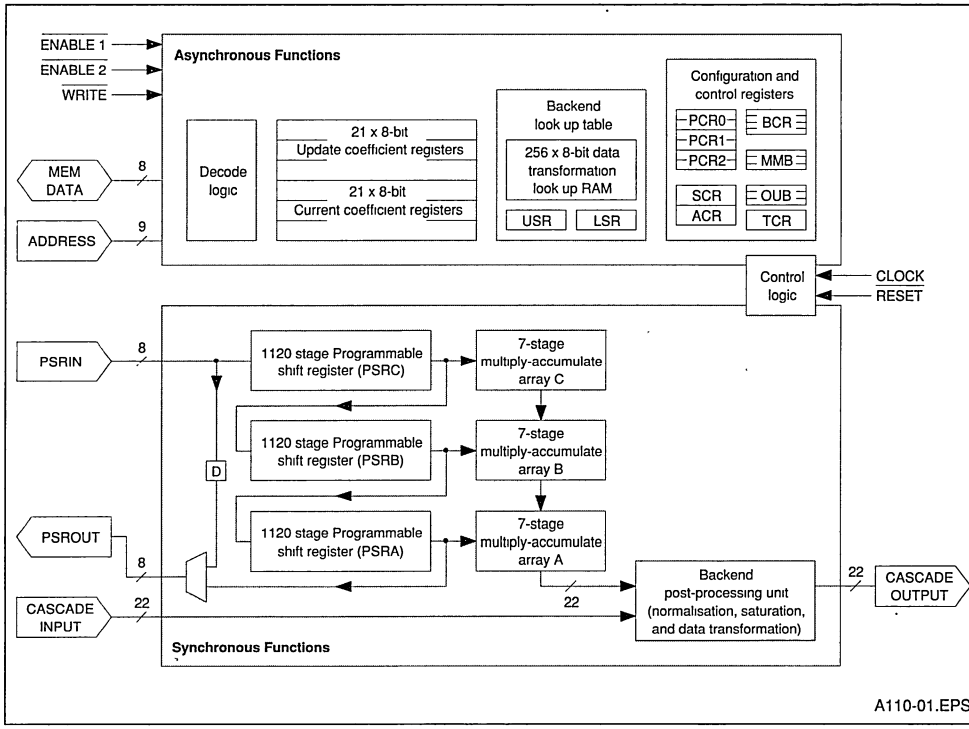
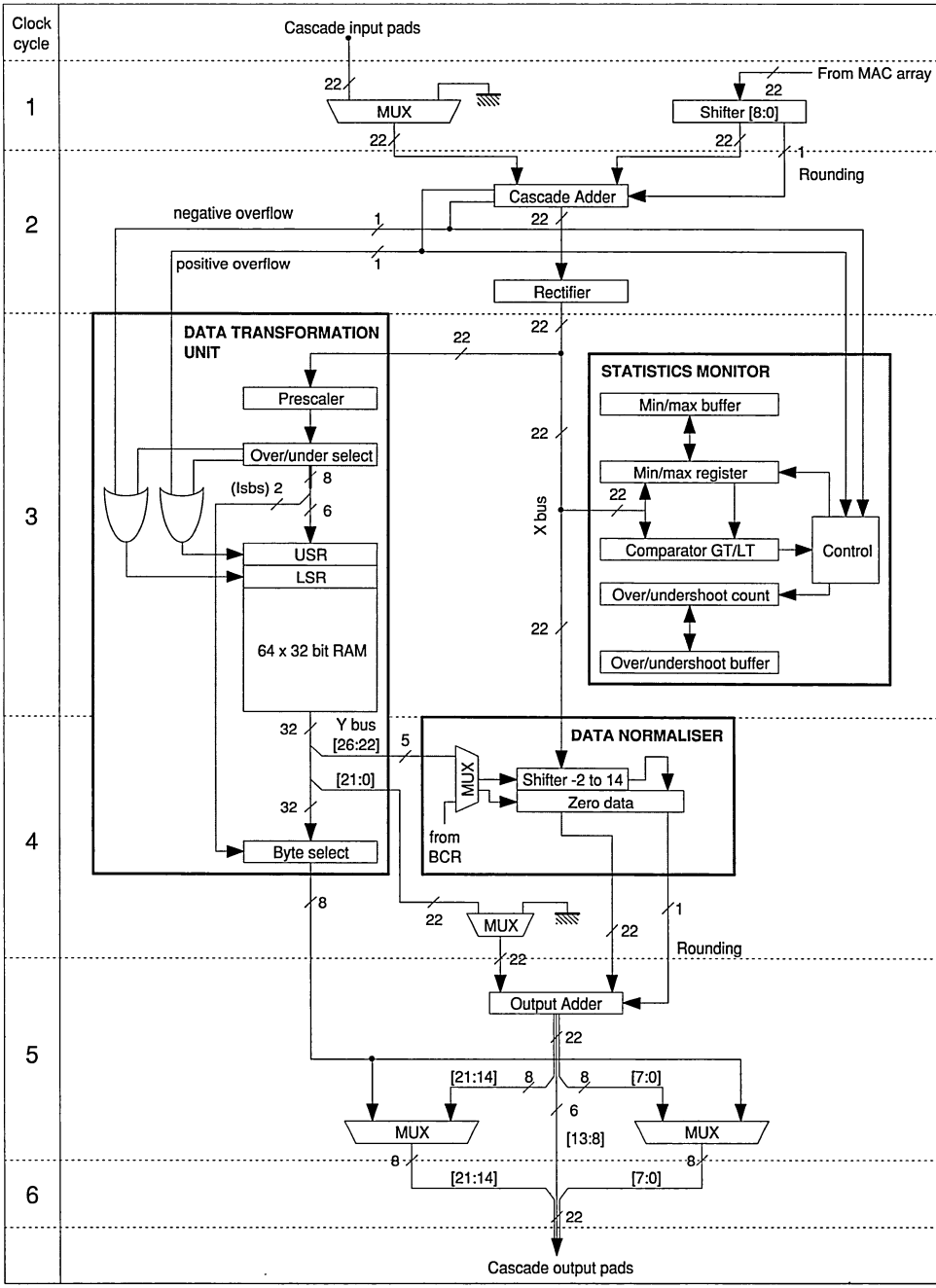


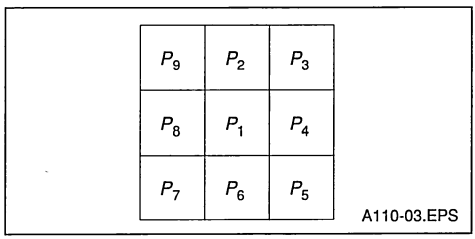
Figure 2 : Detailed Block Diagram of the Back-end Post Processing Unit



2. THE ALGORITHM

Consider an image Im in which every pixel Im(i,j) is either 0 or 1. It is normal for 0 to represent the background and 1 to represent the foreground patterns. It is assumed that each pixel Im(i,j) has eight closest neighbours as shown in Figure 3.

Figure 3 : A Pixel P_1 and Its 8 Closest Neighbours



The output of the algorithm for any given pixel only depends on the value of that pixel and its eight nearest neighbours. This allows parallel processing to be applied with the possibility of all the picture elements being processed simultaneously.

The nature of the algorithm is iterative, each iteration takes the image closer to the fully thinned result. When an iteration is performed which doesn't cause any change to the image then nothing further can be gained by applying further iterations. Each iteration is divided into two subiterations which erode the pattern or patterns to be thinned on opposite edges.

In the first subiteration the pixel P_1 is deleted if all of the following criteria are satisfied:

- $3 \leq B(P_1) \leq 6$
- $A(P_1) = 1$
- $P_2.P_4.P_6 = 0$
- $P_4.P_6.P_8 = 0$

Where $A(P_1)$ is the number of 0 to 1 transitions around the closed path $P_2..P_9$ and $B(P_1)$ is the number of non zero neighbours of P_1.

The second subiteration is identical to the first except that the last two criteria are changed to:

- $P_2.P_4.P_8 = 0$
- $P_2.P_6.P_8 = 0$

It should be noted that this algorithm is not perfect. Some digital patterns will totally disappear. In fact any pattern that can be reduced to a 2 by 2 square will disappear entirely. A solution to this problem will be presented in section 4.

3 DISCRETE IMPLEMENTATION

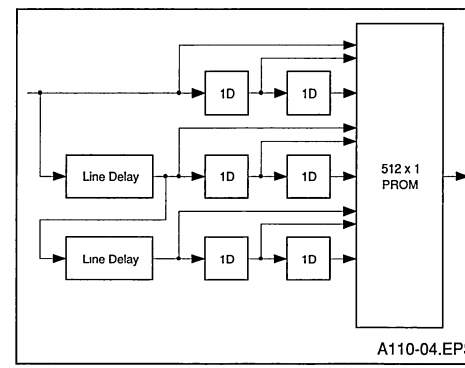
A number of methods of implementing thinning are available. These typically involve the use of arrays of processors such as the ICL DZP (See [5]). Such methods are expensive and physically large but do allow more complicated algorithms than the one presented in the previous section to be implemented.

A simple binary thinning unit which may be built in hardware is shown below. Such a unit is capable of implementing one subeteration of the algorithm described in section 2. It works by arranging for each of the address inputs of the PROM to correspond to one of the pixels in a three pixel square region. By programming the PROM with the appropriate data, which may be calculated from the specified criteria, the output of the PROM gives a new image in which the objects should have been erolded. This process is repeated until no further erosion takes place (note that alternate iterations must use alternate sets of criteria sets of criteria to obtain an unbiased operation).

The performance of such a unit is considerable and it should be fairly trival to process ten million pixels per second. In addition the unit is cascadable which can considerably increase the performance of a system. It does have a number of disadvantages however :

- it is not a single chip solution, unless use is made of semi/full custom chip design.
- it is inflexible. Replacing the PROM with SRAM would improve matters but even so the range of functions such a unit can perform is very limited.

Figure 4 : An Alternative Hardware Implementation

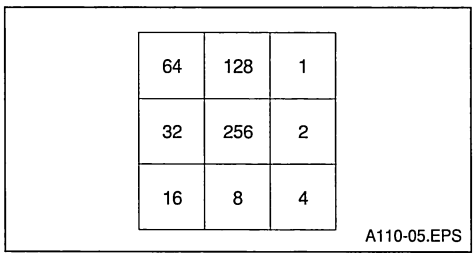


4. A110 IMPLEMENTATION

4.1. The basic implementation

Consider the 2-D filter kernel shown below. When a binary image composed solely of O's and 1's is applied to this kernel the output consists of numbers in the range of 0 to 511. Each output uniquely identifies the pattern of O's and 1's which caused it. By feedit this output into a look up table which has 512 entries it is then possible to generate the output value for P1. The look up table must be programmed with the appropriate pattern of O's and 1's as defined by the criteria for one of the subiterations.

Figure 5 : A Kernel For Binary Thinning



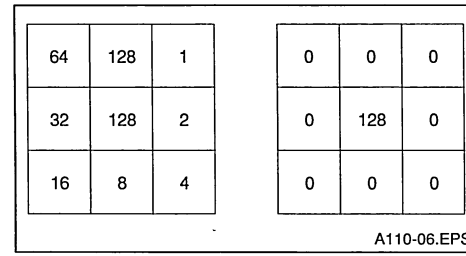
Unfortunately such a kernel cannot be implemented on one IMSA110. There are two reasons for this:

- The LUT only contains 256 entries plus the upper and lower saturation registers.
- Only coefficients between -128 and 255 mays be programmed into the MAC.

The first problem may be overcome by inverting the input image and programming the upper saturation register to 1 so that now any pattern with P1 deleted will give an output which is at least 256. This will cause an overselect to occur at the prescaler. The effect of this is for the LUT output to be taken from the USR (Upper saturation register). Thus the output of the LUT will be 1 which indicates a deleted pixel in the inverted image convention.

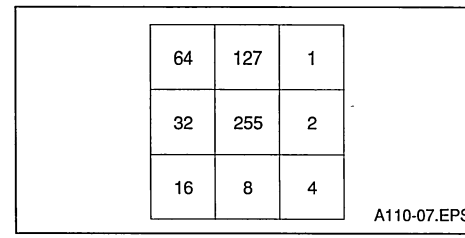
The second problem coulb be overcome by using two IMSA110s. This is achieved by superimposing the two kernels below (See [4] for details about cascading). It would be desirable however to implement each subiteration in a single device.

Figure 6 : Twin Kernels for Binary Thinning



By inspection of the criteria it may be seen that if the coefficients are changed to the kernel shown below then it is still possible to produce a valid look up table. This occurs because although there are two different patterns which can give a MAC output of 127 the required LUT output for each is the same. This method allows a single IMSA110 to fully implement one subiteration of the thinning algorithm. Section 7 shows the data to be programmed into the IMSA110 to implement this algorithm.

Figure 7 : A Kernel For Single IMS A110 Binary Thinning

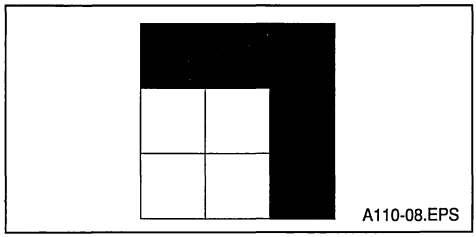


4.2. A solution to the disappearing pattern problem

As mentioned in section 2 any object which thins down to a 2 by 2 pixel square will disappear entirely. This problem may be overcome by slightly modifying the data in the lookup tables.

The first subiteration attacks objects from both below and to the left. In order to stop 2 by 2 pixel regions disappearing it is necessary to stop the elimination of the central pixel in the image segment shown below. Note that white indicates a pixel which is set. o do this the date at location 1F0 in the first subiteration must be set to 0.

Figure 8 : The central pixel in this image



An identical procedure may be applied for the second subiteration except that in this instance the date at location 11F must be set to 0.

4.3. Monitoring the progress of the operation with the statistics monitor

The completion of the thinning operation may be detected by using the statistics monitor in the back-end. The procedure for doing this with a single IMS A110 is as follows:

- Set the max register MMR to 254, and configure the IMSA110 statistics monitor as an overshoot counter.
- Zero the over shoot counter (OUC).
- Perform the first subiteration.
- Record the contents of the OUC register which now indicates the number of pixels already deleted at the start of the iteration.
- Perform the second subiteration.
- Repeat from tep 2 for the next iteration, if the same value is obtained in the OUC register twice in succession then the thinning operation is complete since no further pixels have been deleted.

The actual change in the overshoot count for each iteration may be used as an indication of the amount of progress being made.

5 PERFORMANCE ASSESSMENT

To perform a binary thinning operation on a typical 512 pixel square image using a single IMSA110 would take just over 13 ms for each subiteration at 20 MHz (this neglects the time spent reconfiguring the look up table for the next subiteration). Thus if 12 subiterations were required to fully thin an image then this would take about 156 ms. This performance level is formidable byt mey be easily increased by cascading a number of devices together (See [4]). For example if two devices were cascaded so that the first and second devices performed the first and second subiterations respectively then each complete iteration would still take just over 13ms. So to apply 12 subiterations with this configuration would require only 78ms. This principle may be extended to cascades containing any number of devices (even numbers are preferred since no lookup table reconfiguration is required). If 12 devices were cascaded then the complete thinning operation would take only 13ms. In addition the screen may be sliced up with separate portions being sent to different IMSA110s or cascades of IMSA110s for even greater performance.

The IMSA110 offers other advantages when built into a system since it is capable of performing all the initial image processing commonly associated with image recognition.

- Preprocessing filtering.
- Edge detection.
- Thresholding.
- Thinning.
- Pattern matching.

6 CONCLUSION

It has been shown how the IMSA110 may be used to provide a very high performance and expandable thinning engine. This when coupled to the other abilities of the IMS A10 make the device ideal as a front end processor in many image processing or recognition systems.

7 IMPLEMENTATION DATA

7.1. First subiteration

SCR	090	5C															
ACR	092	00															
CR0a	000	40	7F	01													
CR0b	010	20	FF	02													
CR0c	020	10	08	04													
PCRa	080	Line length +7															
PCRb	082	Line length +7															
PCRc	084	A suitable value to deskew the output image															
BCR0	0A0	01															
BCR1	0A1	00															
BCR2	0A2	40															
BCR3	0A3	80															
USR	0F8	00	00	00	01												
LUT	100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
ACR	092	02															

7.2. Second subiteration

SCR	090	5C															
ACR	092	00															
CR0a	000	40	7F	01													
CR0b	010	20	FF	02													
CR0c	020	10	08	04													
PCRa	080	Line length +7															
PCRb	082	Line length +7															
PCRc	084	A suitable value to deskew the output image															
BCR0	0A0	01															
BCR1	0A1	00															
BCR2	0A2	40															
BCR3	0A3	80															
USR	0F8	00	00	00	01												
LUT	100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
ACR	092	02															

8. REFERENCES

1 T.Y. Zhang, C.Y. Suen _ A fast parallel algorithm for thinning digital patterns. Comm. ACM 27, 3.

2 H.E.Ly, P.S.P. Wang _ A comment on "A fast parallel algorithm for thinning digital patterns". Comm. ACM 29, 3.

3 IMSA110 image and signal processing sub-system datasheet.

4 Cascading IMS A110s application note.

5 C.M. Holt, A. Stewart, M. Clint, R.H. Perrott. An Improved Parallel Thinning Algorithm. Comm. ACM 30, 2.

SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

STi3220
MOTION ESTIMATION PROCESSOR CODEC

By : Marc QUEROL

SUMMARY

This application note, based around three different examples, gives an overview of architectures providing the motion compensation function. More than a collection of schematic diagrams (that would not fit exactly to the user's application), it is more an explanation of what kind of architecture can fit to what kind of application, what precautions must be taken and what kind of components can be used or not.

All the architectures are based around the STi3220 chip developped by SGS-THOMSON microelectronics that provides the motion estimation function. The chip functionnalities will not be detailed here (refer to the STi3220 data sheet for more information): the application note concentrates on the way of providing the good informations to the chip and not on the way of writing or reading those informations into the chip. For that purpose the main part of the note is dedicated to the frame buffer choice and managing.

I - QUICK FEED-BACK ON COMPRESSION TECHNIQUES

Compression techniques trying to reduce the amount of information for the transmission or the storage of a moving picture, are mainly based around the property of pixel's correlation for natural images.

The first technique exploits the spatial correlation of pixels: a pixel in the image has a very high probability of having a value very close to its neighbours average value. The most popular tool using this property is the **Discrete Cosine Transform** which transforms a block of pixels from the original picture into a block of non correlated coefficients. Each coefficient represents a spatial frequency of the original block (the top left one being the average value). Due to high spatial correlation between pixels, for most of the natural images the highest frequency coefficients are of little significance and hence can be reduced or suppressed without altering the picture quality. This is the role of the quantisation block after the DCT.

The second technique exploits the temporal correlation between a pixel in an image and the pixel being at the same position in the previous image. Except when an image and the previous one are totally different (sequence changes), the areas changing from an image to the following are very rare for natural moving pictures. Thus, if **a prediction** of the current image is made, by copying the previous one, the difference between both will most often be close to zero : the non-zero differences represent the moving parts of the picture and are the only information needed to describe the new image from the previous one.

In order to increase the efficiency of the prediction, the **motion estimation** technique is used that associates to a sub-block of the image (also called reference block) a motion vector giving the relative position of the most similar block in the previous image: this block is used as a prediction block. A motion compensation is implemented.

Illustration of the motion compensation technique:

	3 3 2 2
2 2	3 3 2 2
2 2	4 4 3 3
	4 4 3 3
reference block	area of the previous picture centered around the reference block position

If the prediction is made without motion compensation, the predicted block is:

3 2
4 3

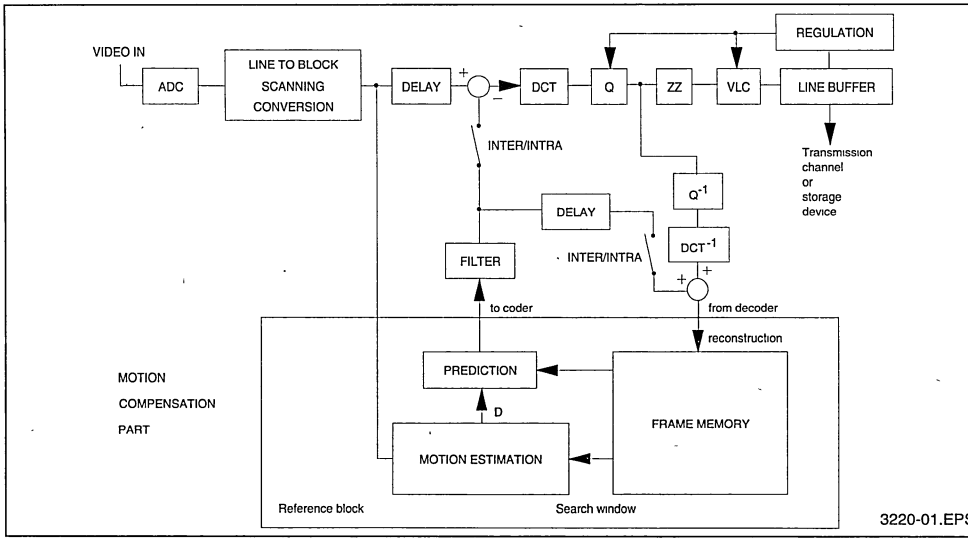
and the difference from the current block is:

1 0
2 1

Using motion estimation will deliver a motion vector equal to +1 in the horizontal direction (East) and -1 in the vertical direction (North). The prediction block is then the same than the reference one and the difference is zero: the only information needed to code the reference block is the motion vector.

The previously described compression techniques lead to the typical moving picture coder scheme shown in Figure 1. This application note only concentrates on the potential implementations of motion compensation.

Figure 1 : Typical Moving Picture Coder Scheme



II - MOTION COMPENSATION

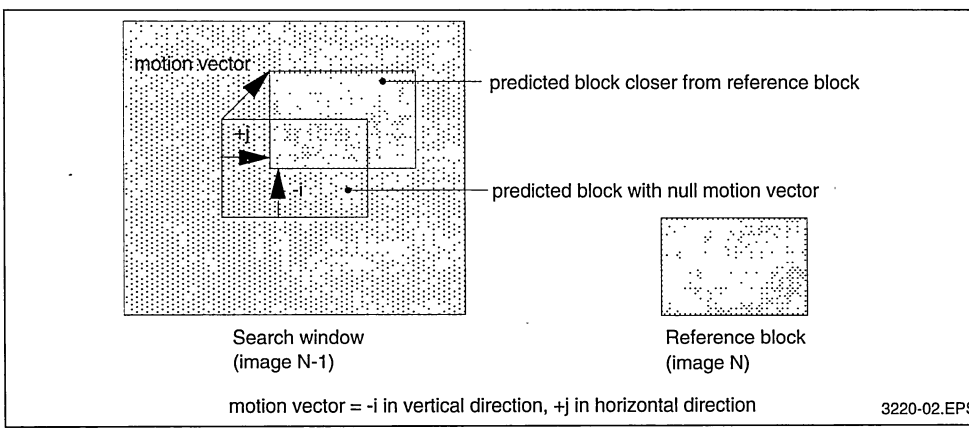
The motion compensation function is organised around three main functional blocks: a frame buffer for storage of the previous image, a motion estimator for research of the best motion vector and a predictor able to deliver to the coder the predicted block pointed to by the motion vector.

II.1 Motion estimation

One of the main devices, needing a lot of computation, is the motion estimator whose role is to find in a limited area of the previous image centered around the reference block position and called search window, the position of the block most similar to the reference block: this position is called the motion vector (see Figure 2).

In most cases, the picture is defined by the three components Y (luminance) U and V (chrominance), but the motion estimation is only made on the luminance component, the same resulting motion vector being applied to all components.

Figure 2



The comparison of the reference block with each possible block of the search window needs a lot of computation. An expression for the distance between the reference block and a predicted block is :

$$D(di,dj) = \sum_{m} \sum_{n} [Ref(m,n) - Pred(m+di,n+dj)]2$$

where :

Ref(m,n) = reference block pixel

Pred(m+di,n+dj) = predictor pixel with displacement di in vertical and dj in horizontal directions.

Computing a distance between two blocks 8x8 for instance needs 64 substractions, 64 multiplications and 64 accumulations. For a -8 to +7 possible displacement in both directions (horizontal and vertical), that is 256 possible predicted blocks, and implies 256 x 64 x 3 = 49152 operations to do during the 64 cycles of the reference block before being able to extract a motion vector.

The STi3220 motion estimation chip, developped by SGS-THOMSON, allows to find a motion vector in the range -8 to +7 in both directions, for reference block sizes of 8 x 4n (from 8x4 to 8x32) or 16 x 4n (from 16x4 to 16x16).

The formula used for computing the distance between the reference block and a candidate block of the search window is the Mean Absolute Error criterion defined by:

$$D(di,dj) = \sum_{m} \sum_{n} |Ref(m,n) - Pred(m+di,n+dj)|.$$

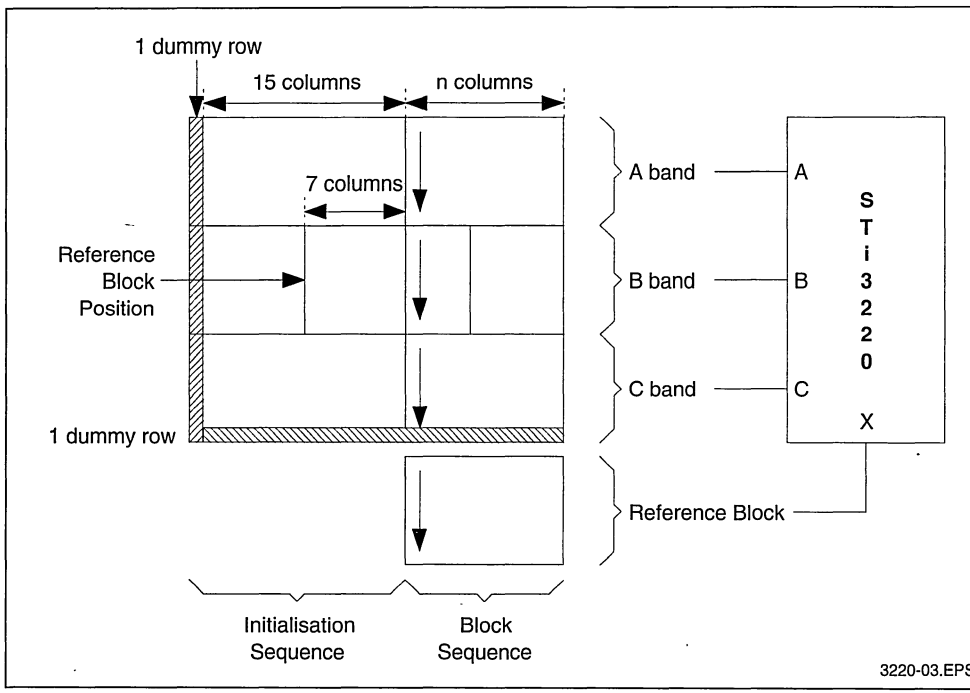
This distance is also called distortion.

The motion vector is the coordinates of the minimum distortion.

In order to compute the distortions, the STi3220 chip must be loaded with the search window and the reference block. For that purpose the chip has 4 input buses : one input for the reference block (X bus), one for the search window band above the reference block band (A bus), one for the search window band corresponding to the reference block one (B bus) and one input for the lower search window band (C bus).

Loading the chip is made column by column from top to bottom and left to right, a pixel being input on each of the four buses at each clock cycle.

Figure 3 : Search Window Input



Loading the chip is done in two phases (see figure 2.2):

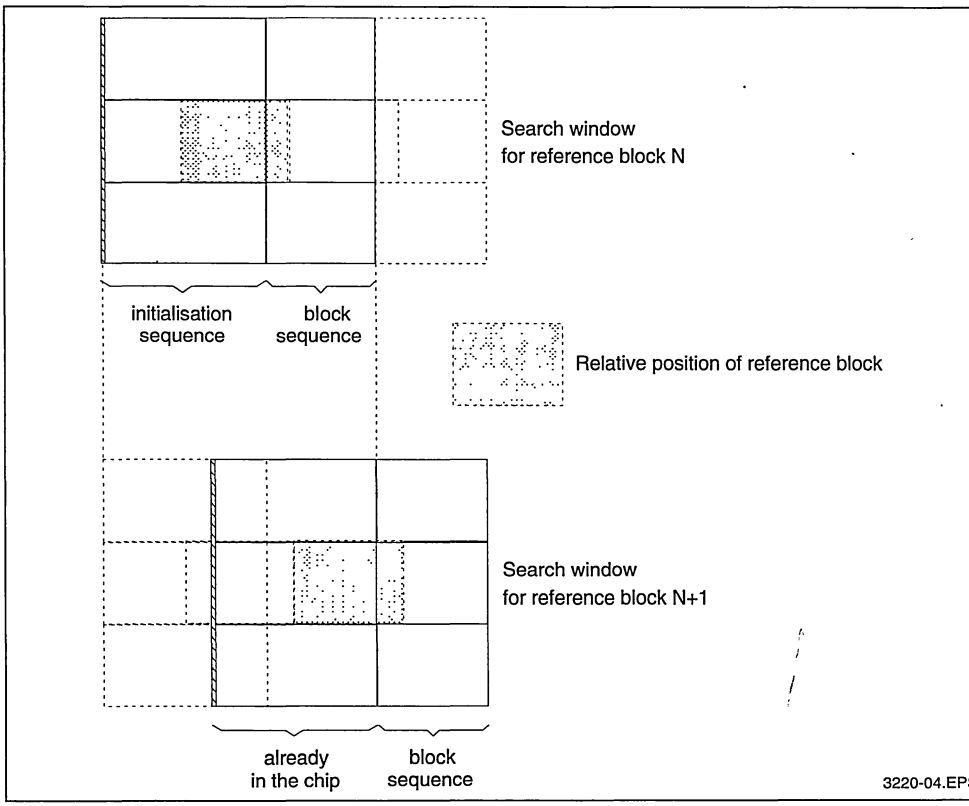
- Initialisation sequence : Input of a dummy column of the search window used to initialise the internal pipeline architecture of the chip, followed by the 15 left-most columns of the search window. During that phase, the reference block input (X) is insignificant. If the reference block is 8-pixel high the initialisation phase costs 8x16=128 cycles. If the height is 16 pixels, the initialisation phase is 256 cycles long.
- Block sequence : During this phase the reference block and the end of the search window are input simultaneously into the chip. The last column of the reference block will exactly correspond to the last column of the search window. The chip will deliver the motion vector 38 (8x4n blocks) or 46 (16x4n blocks) cycles after the last pixel of the reference block, on a specific 8-bit bus IOB.

If the picture is computed in a band by band fashion, then the chip always contains the leftmost part of the search window that will be used for the next reference block and that is a subset of the current reference block's search window. That means that after the very first initialisation phase, a pipeline mode can be implemented: it is only mandatory to input the right part of the search window with the reference block, the left-most part being already in the chip (see Figure 4).

Chapters III and IV show examples of pipeline mode usage with two block sizes (8x8 and 16x16). For motion vectors outside of the range -8 to +7, the search window must be cut into sub-windows supported by the chip (i.e. -8 to +7 max in both directions around the reference block size). The computations must be done:

- by several chips in parallel each one dedicated to a particular sub search window.
- by only one chip computing the motion vector on the different sub windows in several passes.

Figure 4 : Pipeline Mode between Two Consecutive Blocks



Chapter V shows an implementation of -16 to +15 research using one chip.

II.2 Prediction

- After computation of the motion vector for the Y component, it is necessary to extract, for each of the three components, a predicted block that will be subtracted from the reference block for coding. There are two main ways of delivering the predicted blocks :
- the first one consists in accessing directly into the reconstructed frame buffer (see Figure 5). This implies that we have enough bandwith available on the frame buffer ports.
- the second one, is used if the frame buffer bandwith is not sufficient and consists in storing in an additional RAM the search window as it is sent to the STi3220 and to access that RAM for delivery of the predicted block (see Figure 6). In order to be able to deliver the U and V predicted blocks, their corresponding search windows should also be sent to the additional RAM (while the STi3220 chip is disabled: EN input = 1). This implies that the STi3220 needs to be run at a higher speed.
- An alternative could be to split the frame buffer into two fields : one for Y and another for U and V. Y frame buffer is read out for search window delivery (prediction on additional RAM) while U/V frame buffer is accessed for prediction output (see Figure 7).

II.3 Frame buffer

The frame buffer is shared by several resources:

a) write of the reconstructed picture after decoding (necessary on Y, U and V components). As the decoder works on blocks of the image (for DCT) the write will be made in a block by block fashion. If the blocks are delivered by the decoder in a column order they can be directly used in the way they have been stored for the search window delivery.

b) read of the search window for motion estimation (only necessary on Y component). Reading the search window can be made in two ways :

- accessing 3 consecutive times to the frame buffer during one input cycle in order to deliver a pixel for the upper, middle and lower band of search window (see Figure 8). If F is the frequency of the input samples delivered by the line to block scanning, than the frame memory must be read at 3xF for search window access.
- accessing only one time to the frame buffer (one block after each other) and using two delay lines in order to provide the STi3220 with the three necessary search window bands (see Figure 9). The lower band of the search window is directly the output of the frame buffer, the middle band is the output of the first delay line and the upper band is the output of the second delay line. The delay line length is one line of block.

Figure 5 : Prediction access into the frame memory

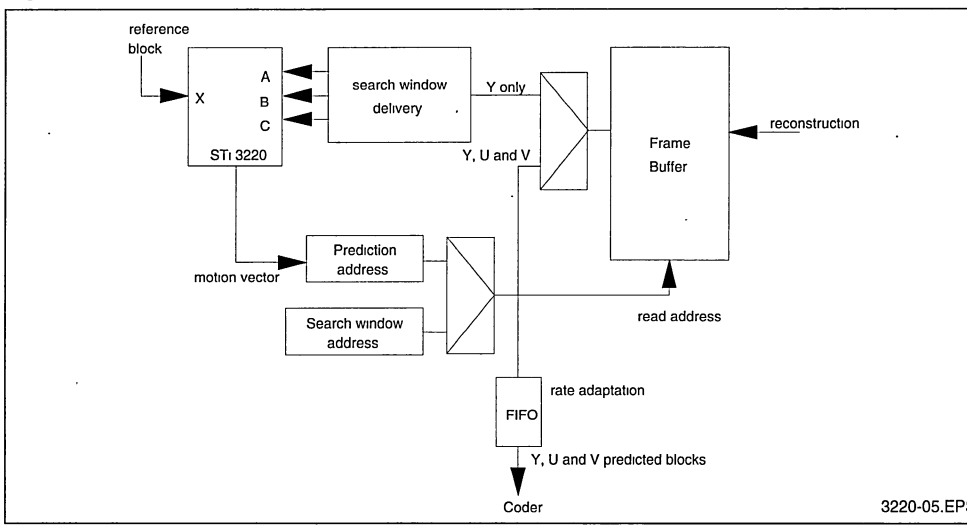


Figure 6 : Predictor Access into Additionnal Ram

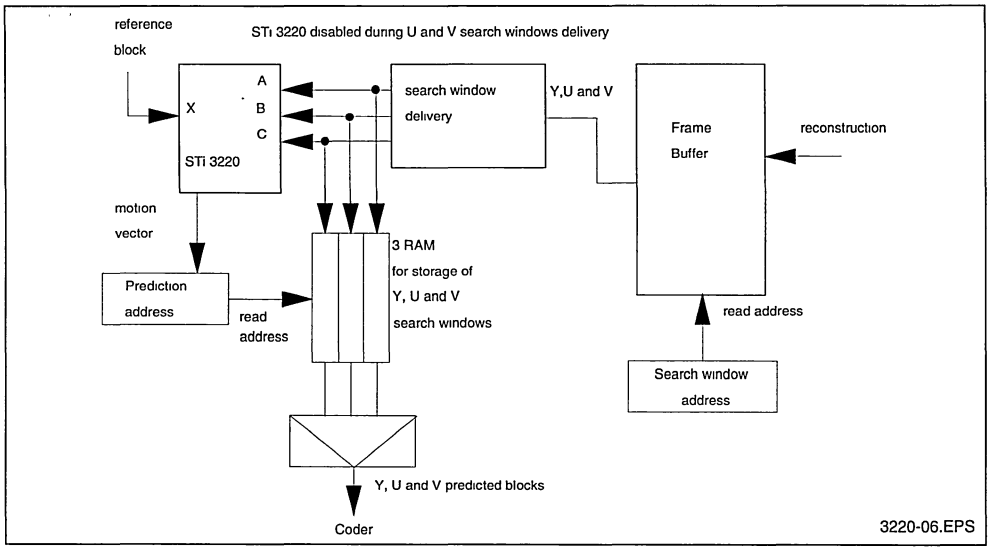
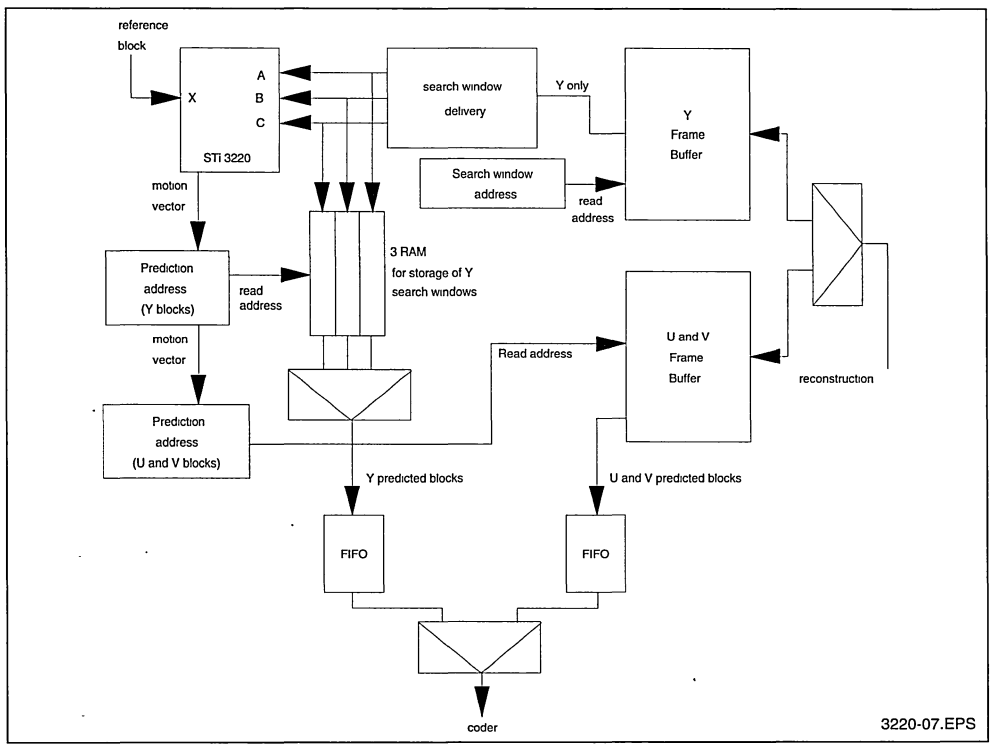


Figure 7 : Predictor Access into AdditionnaL RAM (Y) and Frame Buffer (U and V)



c) If the bandwith is sufficient, the frame buffer may be also accessed for delivery of the predicted Y, U and V blocks. If the bandwith is not high enough, the prediction cannot be made directly into the frame buffer and will be realised in an additional RAM (as explained in paragraph II.2).

The three previous parameters (frame reconstruction, search window and prediction) are the principle ones that will be taken into account in this note, but depending on the application (frame size, Y U V format ...) they can share the frame buffer with other resources. For instance an area of the memory can be reserved for implementation of the line to block conversion. Or the user may want to visualise the content of the frame buffer: in that case at least two frame stores are necessary, one for reconstruction, the other for visualisation. No doubt that other resources of the coder would require memory space and that the frame memory (that is not only used for frame) could provide free areas to use. The main problem will rapidly be the limitation on the memory access times that forces to duplicate the number of memory chips.

Figure 8 : 3 Accesses in Frame Buffer for Search Window

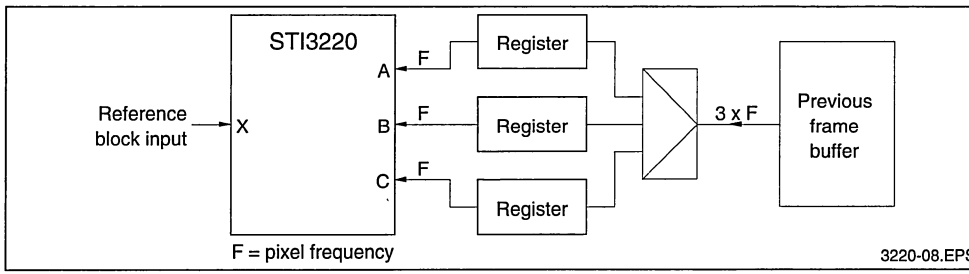
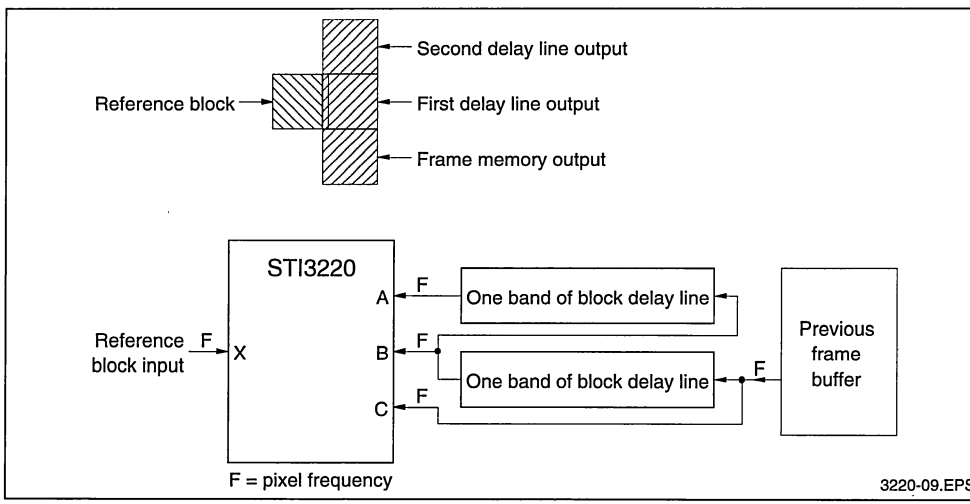


Figure 9 : Delay Lines for Search Window Delivery



III - EXAMPLE 1 : 720 X 576 X 25 HZ PICTURE, 8X8 BLOCKS, -8 TO +7 SEARCH WINDOW

III - 1 Introduction

The first example of a motion compensation implementation is based around a TV picture format as defined by the CCIR 601 recommendation, i.e. :

- picture size: 720 x 576 pixels. The example considers a 25 pictures per second application.
- 4.2.2. format : Picture defined by the three 8-bit components Y (luminance) U and V (chrominance). In 4.2.2. format each pixel is described by

two components : one sample of Y and one sample of U or V. The U and V components are sub-sampled (2:1) in the horizontal direction.

Note : If the picture is originally divided between two interlaced frames, it must be used in a non interlaced form for efficient motion estimation.

The study will concentrate on the motion compensation function considering that the samples are delivered in input in an 8 by 8 block's form (scanned column by column from top to bottom and from left to right) with an alternance of one block of Y and one block of U or V. After the computation delay, the motion compensation function delivers for the coder a predicted block corresponding to the input reference block and waits back from the decoder for the reconstructed block.

The study is made throught two architectures choices :

- the first one is the simplest one. It allows to point out some of the key points of such an application : frame memory choice, delay lines realisation, frame memory organisation and predictor access. The result is a frame address generator easy to realise but a solution that implies the use of very fast memories.
- the second one is a solution where the use of the same amount of frame memory is more optimised and the number of external components is reduced. On the other hand, the address generator is much more complicated to realise.

The reader who doesn't want to follow the different steps of the study and enter into details can directly refer to the architectures block diagrams in Figure 10 and 14 and jump to the conclusion paragraph III.4.

III-2 First architecture proposal

a) Frame memory choice

Pixel rate : 720 x 576 x 25 = 10.368 Mpixel/s. Let's name "f" that frequency. With two components per pixel, the input byte frequency is 2f = 20.736 Mbytes/s.

As the motion estimation is only made on the Y component the samples' rate on the STi3220 can be set to f = 10.368MHz. The U and V samples are directly sent to the coder.

Due to the high samples' rate it seems difficult to read the 3 search window bands directly into the frame buffer at 3f speed. The solution with two delay lines, that allows to reduce the number of accesses on the frame buffer, will be kept for that reason (refer to chapter II.3 for principle).

Frame buffer size = 720 x 576 x 2 = 414,720 x 2 = 829,440 bytes

Necessary bandwith on the frame buffer:

- "2f" for the picture reconstruction (Y and U/V components).
- "f" for reading the search window on Y component.
- "2f" for extracting the predicted blocks on Y and U or V components.

This is a total frequency of "5f" (roughtly 50MHz).

This frequency is too high for a high capacity static RAM or for a classical dynamic RAM.

Dual port dynamic RAM (also called video RAM) **fit well to the application.**

serial port: fast sequential access for reconstruction or search window delivery (regular block by block fashion).

random port: random access for predicted blocks read. But 2 accesses still remain to be done on the random port i.e. a frequency of 20.736MHz : this is too high for the existing components even if using the fast page mode : for instance a 256K x 4 video RAM, with access time of 100ns, has a 55ns read cycle when using fast page mode.

It becomes necessary to **split the RAM into two fields :**

one field for Y and one field for U and V (see Figure 10).

Each field is 411,720 bytes long constituted by four 256Kx4 VRAM. In that way it becomes possible to read in parallel the predicted blocks on Y and U/V frame buffers at "f" frequency. The rate adaptation, delivering alternatively the Y or U/V predicted blocks to the coder at "2f" frequency, is made thanks to two output FIFOs (one FIFO is only 64 bytes long).

The reconstructed blocks are sent back by the decoder to the frame buffer at "2f" frequency and are alternatively Y or U/V blocks. They will be written on the serial ports of each RAM: "2f" frequency is a cycle time of 48.22ns and the minimum write cycle time on the serial port of a 100ns video RAM is 30ns.

An optimisation consists in **writing a block only on one row of the memory** and not among two rows: in that way only one transfer is necessary between the Serial Access Memory (SAM) and the RAM for one block written.

As the reconstructed blocks are written alternatively on Y or U/V frame buffer, each serial port is only used one half of the time. When not used for block reconstruction, the serial port of Y buffer can be used for outputting a block of the search window. This output must be done at "2f" frequency and a rate adaptation is necessary for the STi3220 working at "f" frequency : this is done with a 64-byte FIFO.

b) Memory organisation

Four 256K x 4 video RAMs are used for storing the 414,720 bytes of one frame buffer (Y or U/V). They are organised as a 512x512 nibble array. In order to optimise the use of their serial ports, each block must be stored on only one row of the memory.

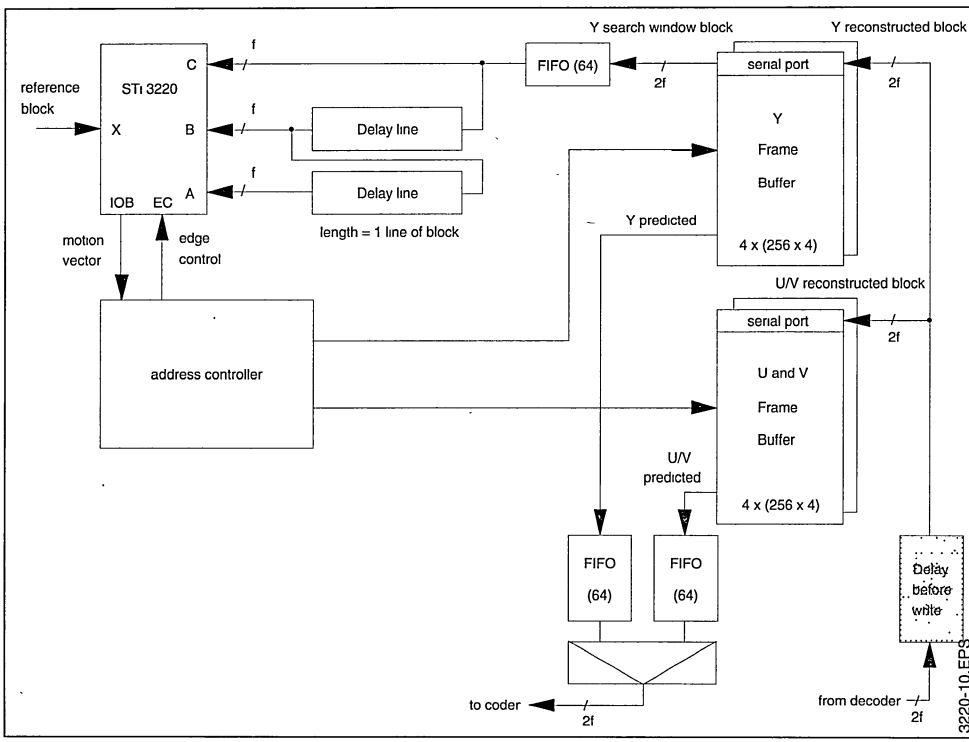
The memory space organisation becomes:

picture						
8	8					
	1	2	3	...	89	90
	91	92	93	...	179	180

frame memory								
	64							
row 0	1	2	3	4	5	6	7	8
row 1	9	10	11	12	13	14	15	16
.	.							
row 10	81	82	83	84	85	86	87	88
row 11	89	90						
row 12	91	92	93	94	...			

The picture format is 720x576 that means 90x72 blocks 8x8. As the 64 samples of a block are all stored on the same row, one row of 512 bytes is able to store 8 consecutive blocks. One line of block is stored on 12 consecutive rows, the last one containing only 2 blocks. With 72 lines of block in an image, the number of rows used in each frame buffer will be 72 x 12 = 864 rows (512 x 2 = 1024 rows available)

Figure 10 : First Architecture Proposal



c) Delay lines

Delay lines are necessary on two different points of this motion compensation architecture :

- a first set of two delay lines, already mentionned, is necessary for providing the STi3220 with the search window. Each delay line is one line of block long i.e. 720x8 = 5760 bytes long. The delay line can be made with an 8Kx8 SRAM associated with an address counter (see Figure 11). Each byte of the SRAM pointed to by the counter is first read out and then written with the new value to input in the delay line (read modify write). When the counter (in fact decrementing) reaches 0 it is re-loaded with the line length value. As two accesses to the RAM must be done during one cycle (1/10.368MHz = 96.45ns) the RAM cycle time must be at least 40ns.
- a second delay line is necessary to delay the reconstruction of the picture into the frame memory. As a matter of fact, the reconstruction of the new picture must not destroy pixels of the old picture that are still necessary, in particular for prediction.

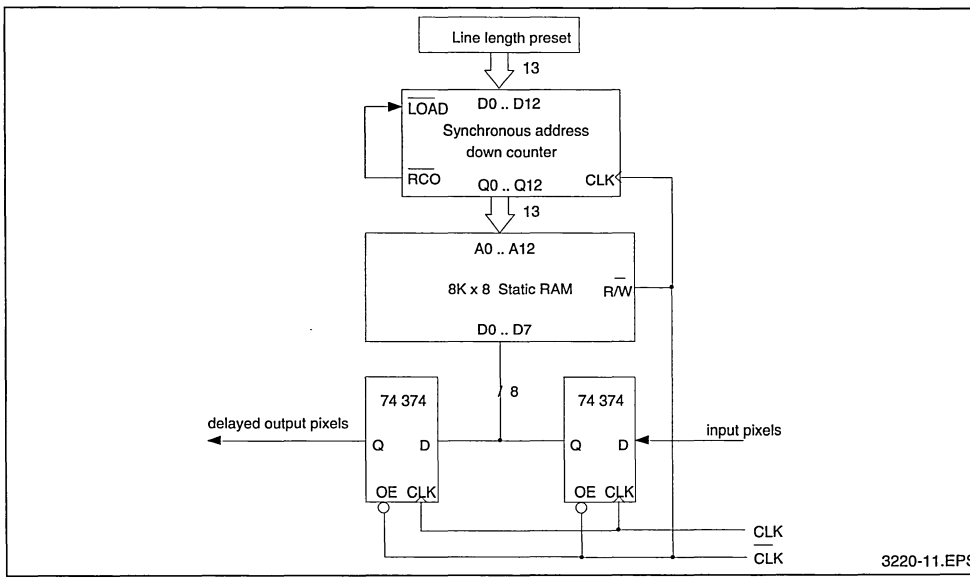
...	B1-1	B1-2	B1-3	B1-4	...
...	B2-1	B2-2	B2-3	B2-4	...
...	B3-1	B3-2	B3-3	B3-4	...

For instance, when the reconstructed block corresponding to reference block position B1-2 is delivered by the decoder to the frame buffer, it is not possible to write this block in the place occupied by B1-2 as long as B1-2 can be used. The last use of B1-2 may happen when delivering the predicted block corresponding to reference block B2-3 as B1-2 is in the search window centered around B2-3.

Therefore the delay between the time a predictor is extracted from the frame memory and the time when it is possible to write the reconstructed block into the memory is one line of blocks plus two blocks i.e. (720 + 2x8) x 8 = 5888 pixels. As each pixel is equivalent to two bytes (Y and U/V), this delay line length must be 11776 bytes (minus the cycles lost outside the motion compensation part for computation in the coder and the decoder).

The delay line can be made with the same principle than the previous ones, but taking care of the fact that the working frequency is "2f", i.e. a cycle time of 48.2ns. To cope with such a rate it could be possible to use two sets of two 16K x 4 static RAM in flip-flop: one set is written while the other is read. This solution is quite expensive.

Figure 11 : Example of Delay Line Implementation

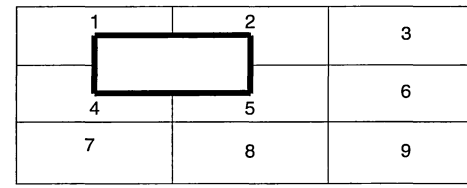


To avoid those external SRAM, the new reconstructed frame (let's call it frame 2) can be stored in the free address just after the last address of the current frame used for prediction (let's call it frame 1) as it can be seen in Figure 12. The only condition to implement such a structure is to have a free area in the memory higher than the requested delay. When the read reaches the end of frame 1, the address generation only continues its increase for starting the read of frame 2: the address generation is cyclic over all the memory space.

d) Predictor access

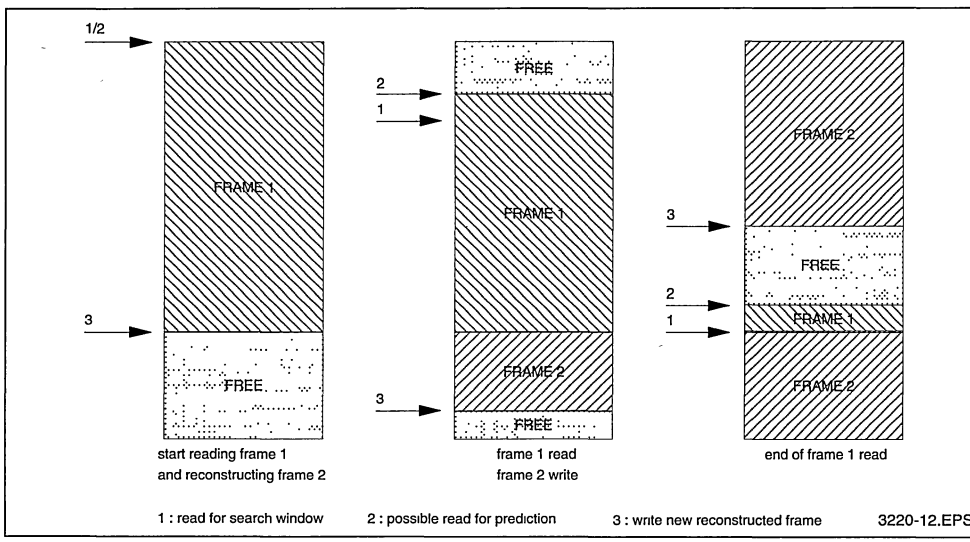
As seen before, reading a predicted block must be done at "f" frequency, i.e. a cycle time of 96.45 ns. However the access time on the random port of a 100ns video RAM is only 190ns. In order to cope with an average access time of 96ns, the page mode access of the video RAM must be used as much as possible : as a matter of fact, the row and column addresses (RAS and CAS selection) must be preset for each random access while for a page mode the row is only selected once (RAS), all the following accesses being done on the same row (also called page) with only a selection of the good column address (CAS). The fast page mode access is only 55ns for a 100ns video RAM like the HITACHI's 256Kx4 HM534251.

The addresses for the predicted block must be calculated depending on the motion vector delivered by the motion estimation chip STi3220. This motion vector is delivered by the STi3220 on the 36th cycle after the end of the reference block (in pipeline mode, it is equivalent to the 36th cycle after the beginning of the next reference block). As said before one block of the picture is not written among two pages of the memory. Nevertheless, except for a null motion vector, the predicted block involves several blocks of the frame memory i.e. several pages. In the example below the predicted block involves pixels from blocks # 1, 2, 4 and 5.



In order to output the predicted block in a column by column order, it is necessary to select a new page twice during each column (access to block #1 followed by block #4 in order to output the first column of the predicted block in the example above). Thus two RAS are necessary for 8 CAS: 2 random accesses and 6 fast page mode accesses. With a 100ns VRAM, reading 8 bytes of a column is 2x190 + 6x55 = 710ns long, i.e. for reading a predicted block : 710ns x 8 = 5.68µs.

Figure 12 : Illustration of Frame Stores in the Memory Space



While reading a predicted block, it is also necessary to manage the serial port of the dual port RAM. A maximum of three transfers between RAM and SAM is necessary :

- one transfer of the reconstructed block written in the SAM to the RAM.
- Two transfers of RAM blocks needed on the SAM for delivery of the search window (the search window blocks are not in the same position than the reference frame blocks but shifted seven columns right (refer to chapter 2), i.e. that they cover 2 frame block's positions. Hence an additional RAM to SAM transfer may be necessary if the two consecutive blocks are not stored on the same page).

The time for reading the 64 samples of a predicted block becomes:

710x8 + 3x190 = 6.25µs. This is equivalent to 6.25/64 = 97.65ns per sample, i.e. a frequency of 10.24 Msample/s that must be compared to f = 10.368MHz.

The margin is not very safe with the components existing on the market at the time this note was written. But there is no doubt that new faster video RAM are or will be available at the time the reader is looking to those lines. For that reason, this solution must be kept in mind.

III-3 Second architecture proposal

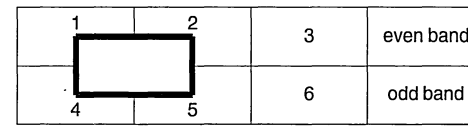
a) prediction

The speed limitation with the previous solution is mainly due to the fact that the page mode of the video RAM cannot be used efficiently when reading the predictor.

As the predicted block most often uses information on 4 blocks of the frame memory, it could be possible to read out all the samples needed for prediction in one of the 4 blocks, then all the needed samples on another block ... The number of random accesses is limited to 4 page changes. But the address generation would be very difficult to manage (no continuity) and an additional output RAM would be necessary in order to rearrange the samples and deliver them to the coder in a column by column order and not in the way they have been extracted from the frame buffer.

Therefore it appears as **a necessity to split again the frame memory in two new parts.**

An interesting cut of the frame memory consists in storing **all the even lines of block in one memory field and all the odd lines in another field.** The predicted block always straddles one even and one odd band of block.



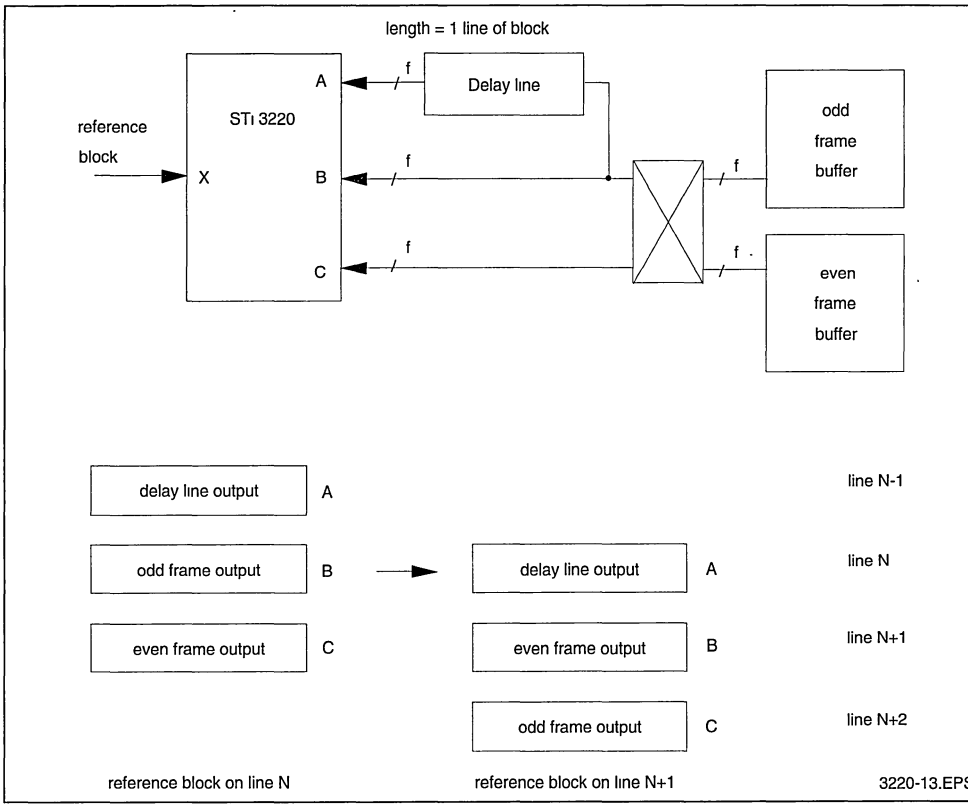
In the example above, block #1 is on even memory field and block #4 on odd memory field. The first columns of the predicted block are selected by a multiplexer between odd and even fields. On each field the selected page doesn't change during all the predicted block output, except if blocks #2 and #5 are not on the same page than blocks #1 and #4 respectively. For reading a predicted block, the maximum number of page changes is limited to two.

Each RAM field becomes an half frame buffer i.e. 207,360 bytes implemented on two 256Kx8 video RAM. The remaining 57,784 bytes of each field is used to begin the storage of the reconstructed picture just after the current one as explained before in "delay lines" chapter.

b) Search window delivery

An interesting consideration of that architecture is that the search window uses simultaneously the odd and even fields of the memory. It is then possible to provide the STi3220 on B and C inputs with the ouptut of the odd and even fields and to generate the A input thanks to a delay line supplied with the B input samples (see Figure 13). This allows to suppress one more delay line replaced by a simple data multiplexer: during one line of blocks, the odd field is connected to B input while the even field is connected to C input. During the following line of blocks, the odd field is connected to C input and the even one to B input.

Figure 13 : STi3220 Supply from 2 Frame Buffers and One Delay Line (example on two consecutive lines)



c) Second architecture timings

Let's verify the reliability of the second architecture, shown in Figure 14 :

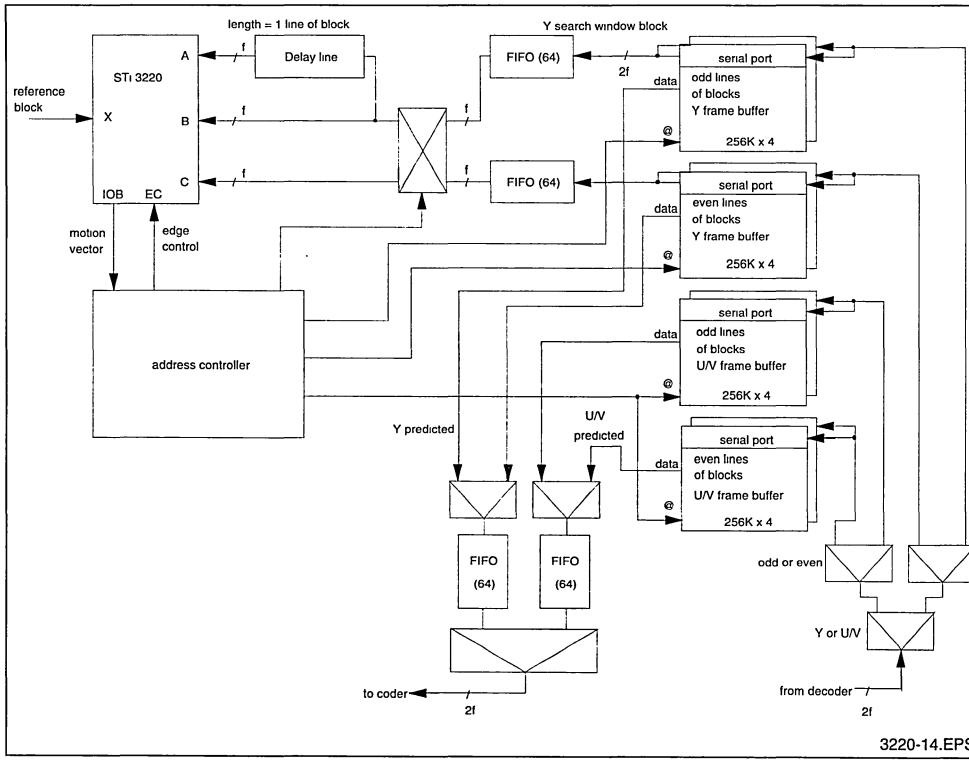
Each memory is accessed on the serial port alternatively for writing a block of the new reconstructed picture (this needs one transfer from SAM to RAM for each block) or for reading the search window in the previous picture (this needs one or two transfers from RAM to SAM). Each transfer between RAM and SAM may cost two random cycles : one for waiting for a current random access on the RAM to be finished and one for transferring the desired row between RAM and SAM. The maximum time lost for two blocks on the serial port is 6 random cycles. Due to transfer cycles, read or write on the serial port must be made at a frequency higher than "2f" .

The random port is accessed for reading the predicted blocks at "f" average frequency. Due to the seperation odd/even lines of blocks, we saw that only two page selections are necessary on each RAM. But it is also necessary to manage the serial memory exchanges with the RAM : as seen before this is a maximum of 3 transfers to do, each transfer followed by a new selection of the page currently read. A total of 8 random accesses may be lost for 5 samples read out, the remaining 59 samples being accessed in fast page mode.

With a random access = 190ns and a fast page mode access = 55ns, the global time for reading a predicted block on the random port is : 8 x 190 + 59 x 55 = 4.765µs. The margin with the time allowed (64/f = 6.17µs) is very safe in that case.

For instance if the fast page mode cycle is fixed to 70ns and the random access to 210 ns (3 x 70) then the total time for reading a predicted block is 8x210 + 59x70 = 5.81µs.

Figure 14 : Second Architecture Proposal



With such clocks, the serial clock cycle can be fixed to 35ns (70ns / 2). The total time for writing a predicted block and reading a search window block is: 128 x 35 + 6 x 210 = 5.74µs compared to the allowed time 6.17µs.

There is no more problem of speed with this solution. Only small FIFOs (64 bytes) are necessary for rate adaptation between the variable samples rate around the frame memory and the fixed computation rate of the coder or the decoder. Of course synchronisation signals indicating the beginning of the blocks will also be necessary.

The time left after the end of a data burst (a block) on the video RAM and the beginning of the next burst will be used for RAM refresh cycles. One refresh can be done simultanously on each video RAM, after each predicted block read sequence, that means one refresh every 6.17µs. This is a total time of 3.16ms for the 512 rows of the video RAMs, compatible with the maximum refresh cycle time of 4ms.

d) Address controller

The address controller, that can be an ASIC or developped around EPLD and PAL devices, must manage several functions:

- SAM : Write of the reconstructed block on the good row and from the good column.
- SAM : Read of the good block for the search window : for instance if the reference block position begins at address n in the even field than the search window access must be done from address n + 7 columns = n + 56 in even field (B input of STI3220) and at address n + 56 (or n + 1 line of block + 56) in odd field (C input of STI3220).

For both read and write the address generation only consists in a counter incrementation. Even or odd field addresses are equal modulo one line of block.

- RAM : read of the predicted block depending on the motion vector delivered by the STI3220 chip on the 36th cycle after the end of the reference block. From the start address of the predicted

block, the address generation consists in a simple counter incrementation plus a selection of the even or the odd field output data.
- generation of the control signals for the RAM chips (including refresh cycles), for the multiplexer selections...

III - 4 Conclusion

This first example has been studied through two architectures quite similar. In both cases eight video RAM chips 256Kx4 are used that allow to separate the sequential access on the serial port from the random access on the parallel port:
- Read search window block on the serial port.
- Write of the reconstructed blocks on the serial port. One block must be stored on one row only to avoid time expensive page changes. The reconstructed frame is stored just after the previous one in order to avoid an external delay line.
- Read of the predicted blocks on the random port. The fast page mode must be used as much as possible to reduce access time. The second solution is much more optimised for reaching that goal.

For such an application the main parameter is the high pixel rate. That was the reason for the choice of search window delivery through delay lines in order to limit the access time on the frame memory. However, if the components are not fast enough, it becomes necessary to split the frame memory into several fields : in that way it becomes possible to read or write simultaneously several informations on several fields at the same time thus reducing the apparent speed in the same ratio. The consequence is that the more the number of fields, the more the number of associated data multiplexers and the more complicated the address generator.

For all those reasons, the first solution, with a memory cut in two fields (Y and U/V frames) is the simplest one (the address generator could be done with counters and PAL devices) but also the solution needing high speed memories. The solution has not enough time margin with the existing components on the market (100ns VRAM), but has been exposed here because the components will go faster and faster or because it can be used for slower pixel rate applications.

The second solution, with a memory cut in four fields (two fields Y and U/V, each one seperated into odd lines of blocks and even lines of blocks) is a better optimisation of the memory use and fits well with the existing components. An intrusting feature of the frame cut allows to suppress one delay line and to replace it by a multiplexer. On the other hand the address generator is more complicated but for a high volume application it can be realised with an ASIC thus providing a solution with a good components' speed/cost tradeoff.

IV - EXAMPLE 2 : CSIF PICTURE, 16X16 BLOCKS, -8 TO +7 SEARCH WINDOW.

IV-1 Introduction

The second example of architecture study is made around a CIF 30Hz format :

frame size = 352 x 288 pixels.
Pixel rate = 3.0413 Mpixel/s.

4.1.1. Format :

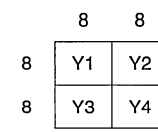
The chrominance components U and V are subsampled (2:1) in both vertical and horizontal directions. Four orthogonal pixels of the picture are defined by four samples of luminance (Y) one sample of chrominance U and one other sample of chrominance V.

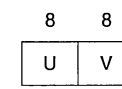
pixels			
Pix O	Pix O	Pix O	Pix O
Pix O	Pix O	Pix O	Pix O
Pix O	Pix O	Pix O	Pix O

associated samples			
YUV O	Y O	YUV O	Y O
Y O	Y O	Y O	Y O
YUV O	Y O	YUV O	Y O

The motion estimation is made on 16x16 blocks of Y, while the coding, and hence the prediction is considered to be made on 8x8 blocks for the three components of the image. Due to sub-sampling ratio, each block 16x16 of Y is associated to one block 8x8 of U component and one block 8x8 of V. The combination of those 3 blocks is also called a macro-block. A CIF picture is composed by 22 x 18 macro blocks.

macro block composition :





One macro block is 256 + 2x64 = 384 bytes.

The samples are considered to be sent to the motion compensation part in the macro block fashion i.e. one block 16x16 of Y scanned column by column (one column is 16 bytes) followed by two blocks 8x8 of U and V also scanned column by column (one column is 8 bytes).

However the predicted blocks must be 8x8 blocks, scanned column by column and sent to the coder in the order:

Y1 Y2 Y3 Y4 U V.

Those blocks after decoding are input back in the same order for the frame reconstruction.

Four pixels of the picture being associated to 4 bytes of luminance and 2 bytes of chromimance, the total frame size is 352 x 288 x 1.5 = 152,064 bytes and the byte rate is 1.5 times higher than the pixel rate i.e. 4.56 Mbyte/s (219ns for one byte). This is the working frequency of coder and decoder.

Considering the slow byte rate of this example, the architecture is chosen with two imperatives:

- **use of classical Dynamic RAM** for frame buffer (e.g. a 100ns access time DRAM has a cycle time of 190ns).
- **access for search window directly in the frame buffer** to supress the external delay lines.

It seams also interesting, for simplification purposes, to manipulate Y and U/V samples in the same flow (no differenciation for search window).

As in the previous example, people in a hurry can directly refer to the architecture in Figure 18 and to the summary in the conclusion chapter IV.3.

IV-2 Architecture choice

a) Search window delivery

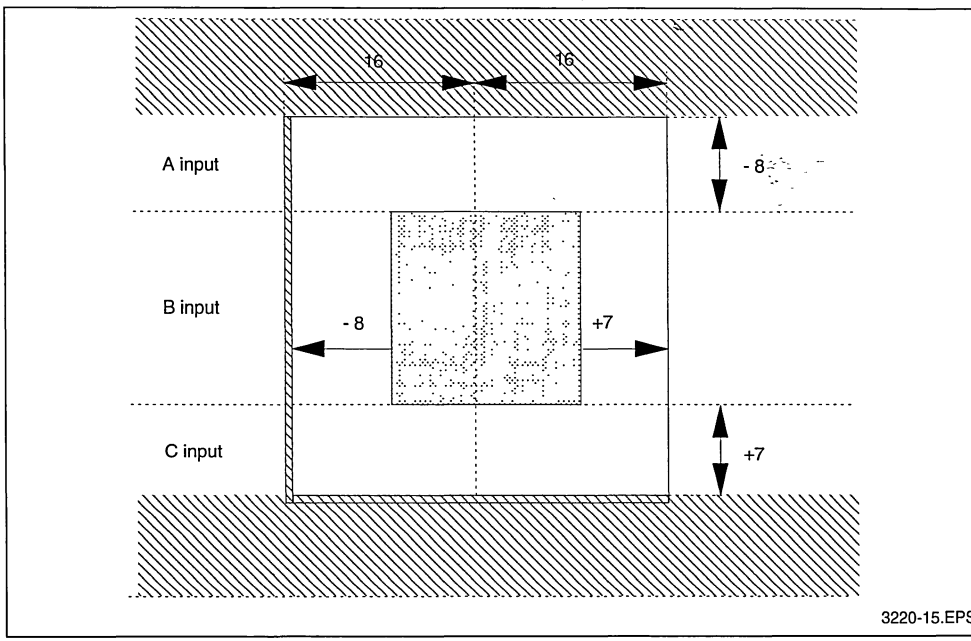
When doing motion estimation on 16x16 blocks with displacement vector in the range -8 to +7, only one half of the upper band of the search window is used (8 lower pixels) and only one half of the lower band is used (7 upper pixels) as shown in Figure 15.

It becomes possible to connect together inputs A and C of the STi3220 as they are not used at the same time.

Sharing the resources between A and C inputs doesn't make the use of two delay lines possible. The search window delivery must directly be made into the frame buffer with two accesses for each reference block sample input.

In order not to double the access frequency on the frame memory, it is possible to split the RAM into two fields : **one field for storage of even lines of macro blocks and one field for storage of odd lines of macro blocks.** In that way only one access is necessary on each field for providing the two search window samples. A data multiplexer will allow the connection of either the even or the odd field to either the B or A+C inputs of the STi3220.

Figure 15 : Search Window and Reference Block Position



b) Prediction

The predictor access will be made in an additionnal static RAM where the search window must be stored on the flight. In order to simplify the system, the U and V components will also be read in the frame buffer for delivery of the search window althought they are not used for motion estimation. When delivering the U or V samples on the A,B and C inputs of the STi3220 it is necessary to disable the chip for not taking care of those information: the EN pin must then be tired to high level. Due to their sub-sampling ratio, the motion vector for U and V components will be half the motion vector of Y component.

The total area necessary for storage of the search window is (see Figure 16):

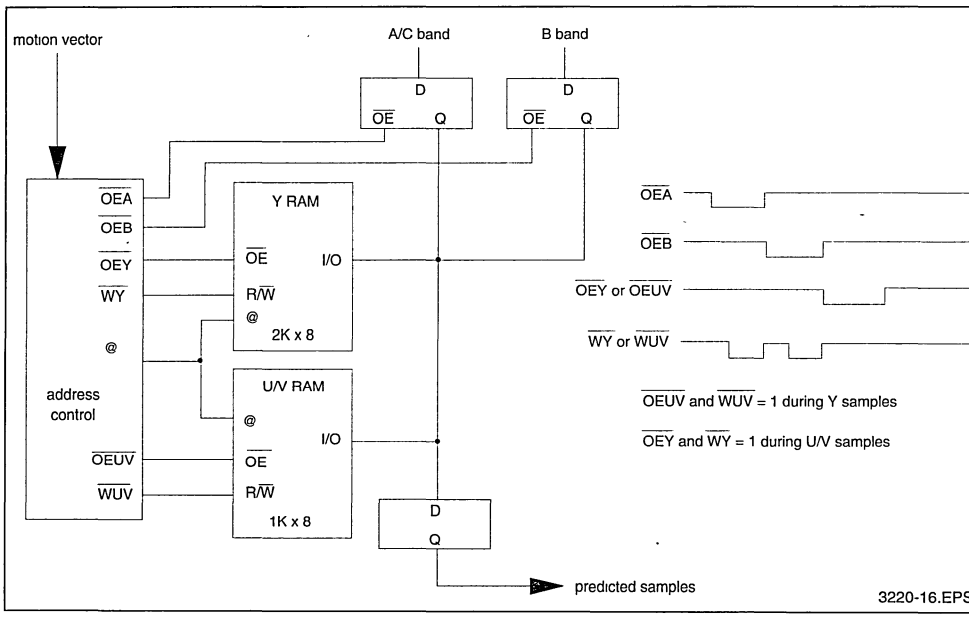
Figure 16 : organisation of the prediction SRAM.

A1	A2	A3	A4	half macro block height
B1	B2	B3	B4	one macro block height
C1	C2	C3	C4	half macro block height

- Four macro blocks for the current search window: 2 macro blocks for B band (B1 and B2) and 2 half macro blocks on either A or C band (respectively A1, A2 and C1, C2).
- One additional macro block on B band (B3) and two half macro blocks on A and C bands (A3 and C3): those macro blocks will be used in conjunction with A2,B2 and C2 for the next search window (let's say the second search window). During the time they are stored, the motion vector corresponding to the first search window and reference block is output by the STi3220 on the 46th cycle. We can consider that the motion vector can be changed by the external system and that it will be taken into account for prediction only with the first sample of the next macro block: that leaves 384 - 46 = 338 cycles for the external system to decide which vector must be used.
- One additionnal macro block on each band (A4,B4,C4) necessary to avoid destroying the first search window read out for prediction with the information of the third search window that must be stored in the SRAM (third search window = A3, A4, B3, B4, C3, C4).

Of course when the predictor of the first search window has been output, locations A1, B1, C1 are free for storage of the last blocks of the fourth search window (fourth search window = A4, A1, B4, B1, C4, C1).

Figure 17 : predictor organisation



The total space needed in the prediction SRAM must be 8 macro blocks i.e. 3072 bytes, that can be divided between one 2Kx8 SRAM for Y component and one 1Kx8 SRAM for U and V (see Figure 17). The SRAM are associated with two latches that temporarely store the samples on each band.

The SRAM is accessed 3 times: one time for storing a sample of one band (B for instance), one time for storing a sample of the other band (C if first half of a column, A if second half) and one time for reading one sample of the predicted block. With a sample rate equal to 219ns, the access time on the SRAM must be at least 73ns.

c) Frame memory choice

The last point to consider is the frame memory that must be shared between frame reconstruction and search window read. As the cycle time of the memory (190ns for a 100ns DRAM) is not far from the byte cycle time (219ns) it is not possible to do two accesses during one cycle : the frame buffer must again be split into two new parts. The samples will be read or written by pair, one memory dedicated to odd samples the other to even samples.

This leads to 4 memory spaces : two fields for storing odd or even lines of blocks and each field constituted by two memories for storing 2 samples in parallel. Each memory must be 38,016 bytes long. The nearest existing DRAM implies the use of 8 DRAM 64K x 4 with access time not higher than 100ns to cope with the 219ns cycle time.

The proposed architecture is shown in Figure 18.

The frame memories are associated to 6 latches. As a matter of fact four samples are read-out from the two fields of two memories during one cycle (needing four latches for temporal storage) and two samples are written in one field of two memories during the following cycle (needing two latches plus a multiplexer for selection of the good field).

d) Memory organisation and refresh

The 64Kx8 DRAM are organised as a 256 x 256 array.

One macro block is 384 bytes shared between two DRAM : so each half macro block of 192 bytes is stored over 2 DRAM.

The picture contains 22 x 18 = 396 macro blocks shared between two fields of DRAM , so each DRAM contains 198 half macro blocks of 192 bytes.

If each half macro block is written on a column of the DRAW (198 columns used in each RAM), each time a macro block is accessed (and there is always a macro block read in each DRAM for search window delivery), the corresponding 192 rows are addressed and hence refreshed. The total time needed for reading a macro block is 384 x 219ns = 84.096µs. This is also the cycle time for refreshing all the useful rows of the memory : it is compatible with the maximum refresh cycle times of 4ms defined for the 64K x 8 DRAM.

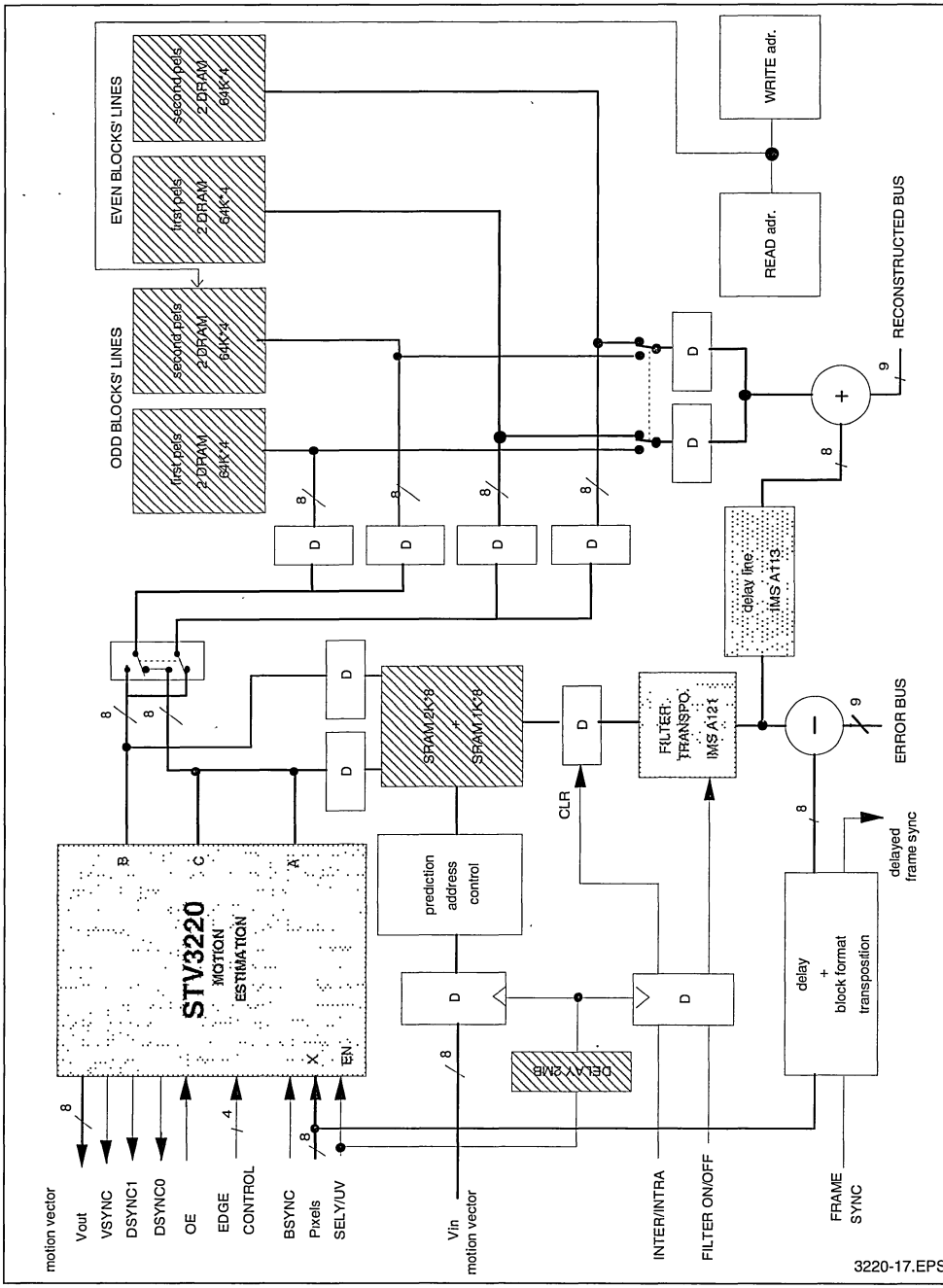
Macro block n :

y0	y1	y3	...	y240
y1	y17	y33	...	y241
y2	y18			
y3	y19			.
y4	y20			.
y5	y21			.
...				.
y14	y30			
y15	y31		...	y255

u0	u8	...	u56
u1	u9	...	u57
u2			
u3			.
u4			.
u5			.
u6			
u7	u15	...	u63

v0	v8	...	v56
v1	v9	...	v57
v2			
v3			
v4			
v5			.
v6			.
v7		...	v63

Figure 18 : Example 2 General Architecture



Memory organisation for storage of macro block n in odd or even field :

	DRAM1 - column n	DRAM2 - column n
row 0	y0	y1
row 1	y2	y
row 2	y4	y5
...		
row 7	y14	y15
row 8	y16	...
...		
row 127	Y254	y255
row 128	u0	u1
row 129	u2	u3
...		
row 159	u62	u63
row 160	v0	v1
row 161	V2	v3
...		
row 191	v62	v63

Rows 192 to 255 are not used. Macro block n+1 is stored on column n+1.

Note: The two memories in each field are always accessed with the same address.

e) Address control

In order to simplify read , the write address generation should take care of the fact that the Y samples are sent back by the decoder in 8x8 blocks (after inverse DCT) and that they will be read in 16x16 blocks for motion estimation: the samples should be stored in a way such as the read only consists in increasing the row addresses. Of course the U and V 8x8 blocks are stored in the way they are coming from the decoder just after the Y block in the memory.

When a memory field is dedicated to the middle band of the search window (B input), the column address doesn't change during one macro block. When the field is dedicated to lower and upper bands (A and C inputs) the 8 last bytes of a column are read out before the 8 first ones (for Y) so it is necessary to change the column address every 8 samples (for Y) or 4 samples (for U or V) but with a row address always increasing from the beginning to the end of the macro block.

f) Filter

In the architecture scheme shown in Figure 18 there is a filter on the predictor output : this filter can be used to increase the efficiency of the prediction by reducing artefacts due to high frequencies. It is realised with an IMS121 device from SGS-THOMSON that is able to perform either Discrete Cosine Transform, Inverse DCT, filtering or matrix transposition. In that case the filter is used that associates with the input predicted block a low-pass filtered output block delivered in a transposed order. Even when the filter is not used in the loop, the IMS121 must be used to do transposition of the predicted blocks in order that they can always remain with the same scanning order in the output stage.

Note: the filtering is done on 8x8 blocks and not 16x16 : that is one of the reasons why the prediction must be made on 8x8 blocks. The delay for the filter or transposition function is 128 cycles.

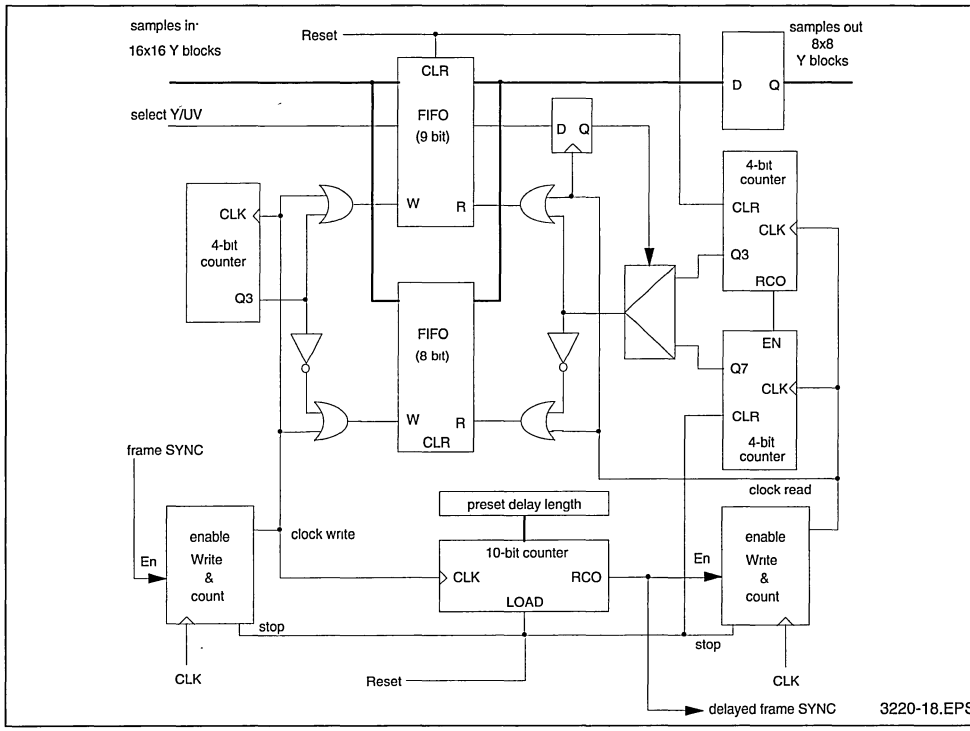
g) Error blocks = reference - prediction

The filtered predicted blocks must be subtracted from their corresponding reference block to provide the error blocks, sampled over 9 bits, that will be computed by the coder (for instance throught DCT). Those blocks must be 8x8 blocks. The transformation from 16x16 to 8x8 blocks is already done for the predicted blocks when reading the search window RAM, but it must be made for the reference block that was sent to the STi3220 in 16x16 form.

The delay between the time a reference block is input in the STi3220 chip and the time it is subtracted from the predicted block is 2 macro blocks for reading the predicted blocks and 128 cycles for filtering i.e. 896 cycles. Of course this is an approximation as the number of cycles depends on the final schematic diagram.

The proposed architecture for delaying the input reference blocks and providing the block format conversion is shown in Figure 19.

Figure 19 : Block Delay + Format Change



When starting, the write on the FIFOs is disabled until the first frame synchronisation appears (with the first pixel). Read on the FIFOs is prohibited until the delayed frame sync. (generated by the counters preloaded to the delay length = 896 cycles) is not delivered.

In write mode the first 8 samples of a column are written on the first FIFO, the 8 following on the second FIFO : the first FIFO will always contain the upper 8x8 blocks of a 16x16 block and the second FIFO the lower ones. In read mode the first FIFO is read 128 consecutive cycles for delivery of the two first 8x8 blocks, and than the second FIFO during 128 cycles for the two following blocks. For U and V blocks (already in 8x8 format) the FIFOs are read in the way they are written.

h) delay predicted blocks

The filtered predicted blocks must not be lost after delivery of the error blocks as they will be used for frame reconstruction in addition with the reconstructed blocks (after decoding). For that purpose they must be delayed during the time the error blocks are computed in the coder and sent back to the board by the decoder. This delay can be made using the SGS-THOMSON A113 chip: this device is a delay line whose length can be programmed from 8 to 1320 steps.

IV-3 Conclusion

The most interesting feature of this architecture is the use of very popular (and hence cheap) components like the 64Kx4 DRAM or the small 1Kx8 or 2Kx8 SRAMs.

Like in the previous example, in order that the DRAM cycle time (190ns for a 100ns DRAM) fit with the input byte cycle time (219 ns) a split of the frame buffer in several fields is necessary:

- one odd lines of blocks field and one even lines of blocks field. On each cycle, one sample is read out from each of the two fields providing the two samples necessary for the search window supply.
- each of the two fields is divided in two parts: in that way, during one cycle two bytes are read out in parallel from each field (one byte in each

sub-part) and during the following cycle two bytes are written in parallel in one of the fields, for frame reconstruction.

The total frame memmory is composed by eight 64Kx4 DRAM.

The frame buffer is only used for search window delivery and for frame reconstruction. The prediction is made in an additional RAM where the search window is stored on the flight as it is sent to the STi3220 inputs. In order to simplify data control, the U and V samples are computed in the same flow than the Y samples. The additional RAM is a 2Kx8 SRAM for Y search window and 1Kx8 for U/V search window.

As the prediction is seperated from the frame buffer, the address generator of the frame memory is greatly simplified and could be realised with several counters and PAL devices.

This architecture is the simplest and the cheapest of the three examples of this application note. Consequently, it is also the architecture that is the fastest to implement with standard components for prototyping a codec. For high volumes, the memory address generator and the predictor could be integrated into a ASIC device leading to a block diagram with the STi3220, the DRAMs and the control ASIC as main devices.

V - EXAMPLE 3 : CIF PICTURE, 16x16 BLOCKS, -16 TO +15 SEARCH WINDOW.

V-1 Introduction

The picture format considered in this example is exactly the same than the previous one :

30 frames/s, 352 x 288 pixels/frame, 12bit/pixel (4.1.1 format): frame size = 152,064 bytes.

But in this example we want to compute a motion vector in the range -16 to +15 in both directions.

The study first explains how to compute such a search window with only one STi3220 able to cover a -8 to +7 range, then a proposal of architecture is made (refer to figure 5.2) followed by an explanation of the search window supply mechanism. Finally, like in the other examples the study ends with a verification of the different accesses that must share the frmae memory bandwith. A conclusion chapter (5-6) is available for people who just want a quick overview.

V-2 Search window computation

a) Search window definition

As the STi3220 chip is only able to compute motion vectors in the range -8 to +7, four different computations over four sub search windows are necessary to cover the range -16 to +15. The sub windows selection is shown in Figure 20.

The four resulting motion vectors are partial ones. In order to determine what will be the final motion vector for the total search window, the minimum distortions, corresponding to each partial motion vector for the total search window, the minimum distortions, corresponding to each partial motion vector, must all be compared. No particular algorithm will be given here to determine what must be the final motion vector as it is a part of the coding policy.

The motion vector can be directly associated to the minimum of the four distortions : for instance, if the minimum is obtained in the sub search window 2, then the resulting motion vector (MV) is the partial motion vector 2 (MV2) plus 8 in horizontal direction and minus 8 in vertical direction.

b) How to compute sub-search windows

The four different calculations for each sub search window can be implemented in two different ways :

- using four STi3220 chips in parallel each one working on a different sub search window. After an initialisation sequence and a block sequence on each STi3220 (i.e. 512 cycles), the computation can be done and the partial motion vectors obtained after 46 new cycles on each chip. This solution is only useful in high speed systems and is an heavy and expensive solution. In an application like the CIF format it is cheaper to use the second solution.
- using only one STi3220 doing all the partial computations on the four sub search windows. Each time a sub search window is computed, an initialisation sequence and a block sequence are necessary. It can be noticed on Figure 20 that the initialisation sequence of sub search windows 2 and 4 are the same as the block sequence of sub search windows 1 and 3 respectively. Hence the pipeline mode can be used between sub search windows 1 and 2 (or 3 and 4). Sending all the necessary data to the chip will cost 6 blocks 16x16 i.e. 1536 cycles. Each time the chip is in a block sequence, the 16x16 reference block must be input on the X bus for comparison with the sub search window. Of course the results of each sub search window, obtained on the 46th cycle following a block sequence, must be stored in order to be analysed at the end of the process (at least the motion vector and the minimum distortion must be kept).

All the sub window scanning must be done during the time of one input reference block. As six sequences are necessary the chip must work at a speed 6 times higher than the Y reference input rate.
Considering the pixel rate = 352x288x30 = 3.041MHz, the chip must run at 18.25MHz i.e. at its maximum working frequency. It implies that Y and U/V samples cannot be mixed on the same flow as the speed would be 1.5 times higher.

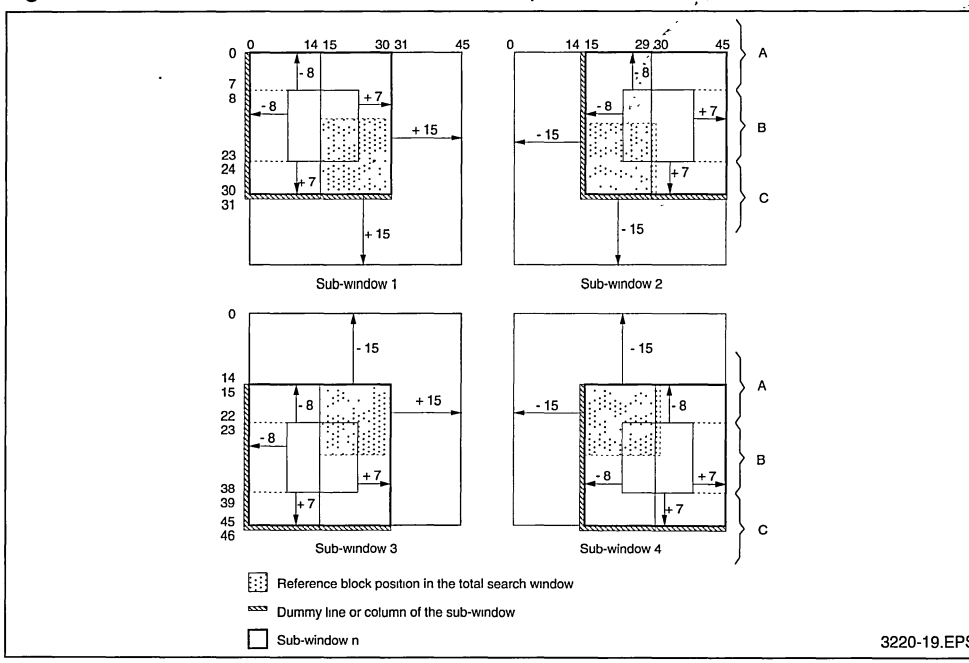
V-3 Architecture study

Due to the division of the search window in four different parts it is not possible to use the delay lines structure in this application.
It could be possible to split the frame memory in two parts (odd and even lines of blocks) in order to access in parallel to the two samples needed for the search window. This implies the use of video RAM in order to cope with the samples rate (about 55ns) but this would lead to memory sizes of 152,064 / 2 = 76,032 bytes which is not optimum with the existing component sizes (64K x 4 or 256K x 4). Hence we will consider a solution with only one frame buffer constituted by two 256K x 4 video RAM memories.

It is not possible to output from the frame buffer all the blocks needed for the search window during the time of only one reference block : the frequency would be 12 x f if f is the pixel's frequency.
A small static RAM will be added to the main frame buffer in order to store the search window : in that way, for each new reference block in the pipeline, only the three new right-most blocks of the search window are necessary (see Figure 20). As it is necessary to provide the search window inputs of the STi3220 with two samples on each cycle, the SRAM is in fact separated in two parts : one SRAM for storing the middle band's search window blocks and one SRAM for storing the upper and lower bands of the search window (as they are not accessed at the same time). The transfer of the new blocks from the frame buffer to the two SRAM will be made throught two FIFOs for rate adaptation.

The last point to consider is that all the SRAM data are read out in the same way they are written: this property induces to use FIFOs instead of SRAM in order to simplify the external control (no counters for address generation).
An architecture able to support this example is shown in Figure 21 and is going to be analysed in the following lines.

Figure 20 : Four Sub Search Window for -16 to +15 Displacement



V-4 Search window delivery

Note: For each block of the search window, the 8 last pixels of each column are sent to the STi3220 chip on B and C buses, followed by the 8 first columns sent on buses A and B (refer to Figure 20). In order to provide the search window blocks in the good order, the output of the frame memory is delayed by 16 cycles (refer to Figure 21): the 8 first samples of a column are first written into the 16 cycles delay device, then the 8 last samples of the column are read out from the memory for search window delivery (they are also written in the delay line). During the 8 following cycles the 8 first samples of the next column are read out from the memory and stored into the delay line while the 8 first samples of the current column are extracted from the delay line for search window.... The delay for delivering the search window is equivalent to 8 cycles.

Let's try to analyse the different phases of the search window delivery, with reference to Figure 22.

Note: The blocks noticed in Figure 22 have not the same borders than the blocks of the image: as a matter of fact their relative position is shifted one column left. The information will be extracted from the frame buffer blocks after blocks but it is easy, with the rate adaptation provided by FIFOS 1A and 1B, to read out the blocks for search window with a relative position anticipated by one column. For simplification purpose we will refer in the explanations to the search window blocks and not to the image blocks keeping in mind that one search window block = one column of a block image N and fifteen columns of block image N+1.

Figure 21 : Example 3 Architecture

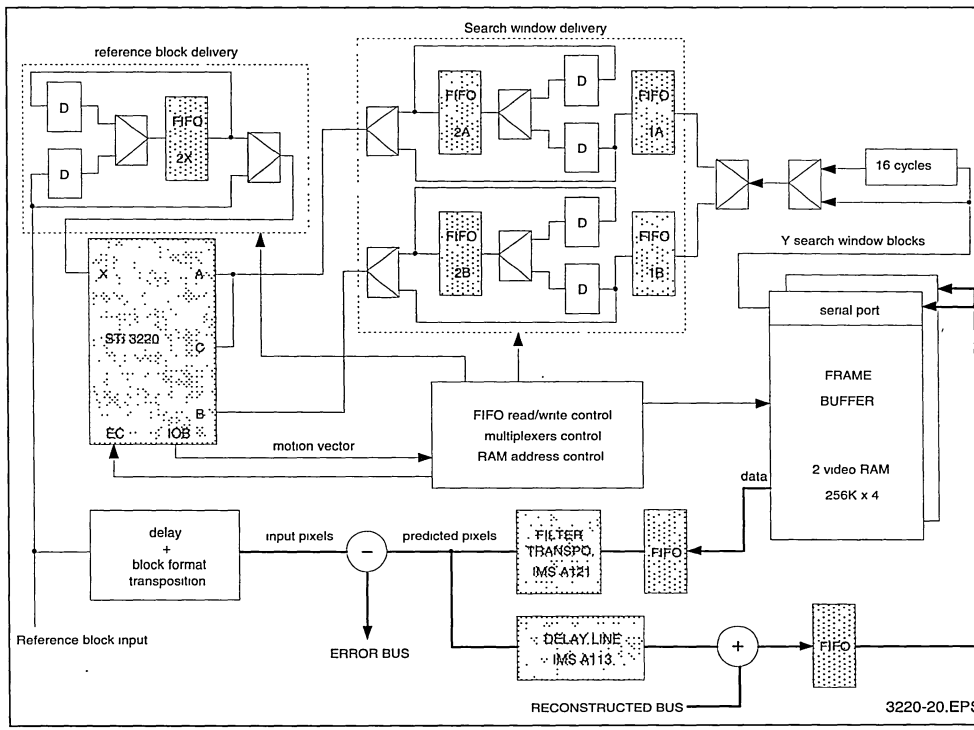
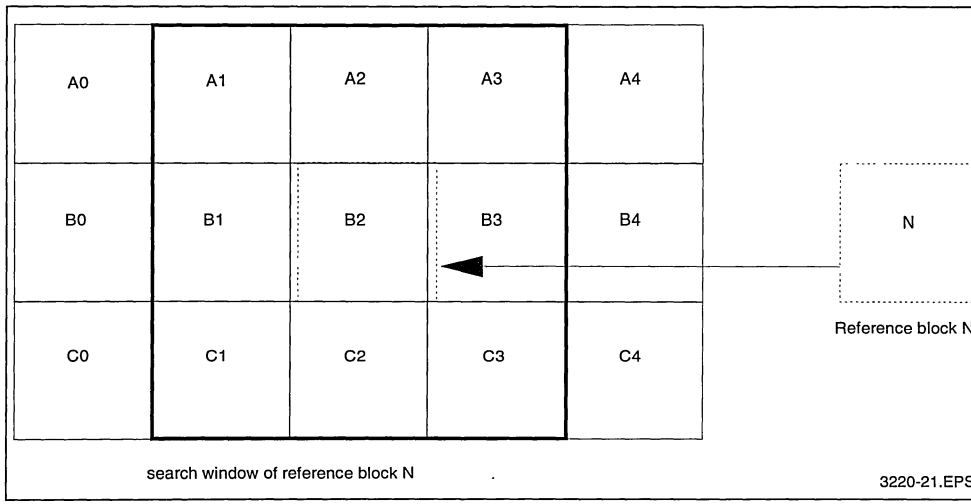


Figure 22 : Search Window Block Notation



The sequences on the STi3220 inputs for reference block N must be:

A/C input :	A1	A2	A3	C1	C2	C3
B input :	B1	B2	B3	B1	B2	B3
X input :	..	N	N	..	N	N

The input sequences for the next reference block N+1 will be :

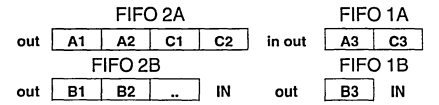
A/C input :	A2	A3	A4	C2	C3	C4
B input :	B2	B3	B4	B2	B3	B4
X input :	..	N+1	N+1	..	N+1	N+1

As it can be seen on those two reference block sequences some blocks of the search window are used several times either for the same search window or for the next one. Hence when outputting a block from the FIFOs it may be necessary to write it back into the same FIFO: the output of the FIFOs 2A and 2B may be connected to their own input.

Search window processing:

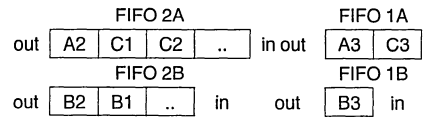
Note: The FIFOs name used in this description can be referred to in figure 5.2.

- At the beginning of the process, we suppose that FIFO 2A (upper and lower bands FIFO) already contains A1 A2 C1 and C2 and that FIFO 2B (middle band) contains B1 and B2. FIFOs 1A and 1B should contain at the beginning respectively A3 C3 and B3. This can be represented by :



There is a free area in FIFO 2B which size is one block.

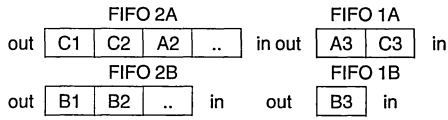
- The first phase is an initialisation phase of the STi3220 and consists in loading the first half part of the first sub search window into the chip. Blocks A1 and B1 are read out from FIFOs 2A and 2B. Block A1 will not be used in another phase so FIFO 2A is not written during that time. On the other hand, block B1 will be used later and hence the block read out from the FIFO 2B is written back into the same FIFO. The reference block input X on the STi3220 is irrelevant during that phase. The FIFOs content at the end of this phase is:



- During the second phase A2 and B2 are output from the FIFOs. This is a block sequence for the STi3220 during which the reference block must also be input. At the end of that phase the first

sub search window is input and the first motion vector computation is running. As blocks A2 and B2 will be used during later phases they must be written back in both FIFOS.

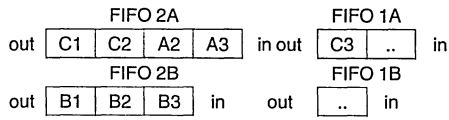
FIFOS content at the end of the phase :



the 3rd phase is also a block sequence for the STi3220 as the pipeline mode can be used between first sub-window and second one. The reference block must again be sent to the STi3220. During the same time blocks A3 and B3 are needed to finish the second search window: as those blocks are not yet into FIFOS 2A and 2B, they are read out from FIFOs 1A and 1B while they are also stored into FIFOs 2A and 2B in their free locations i.e. just after blocks A2 and B2. FIFOs 2A and 2B are not read during that phase.

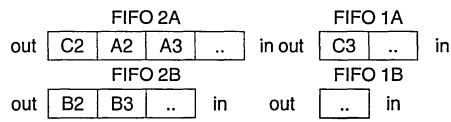
Note: on the 46th to 49th cycle of this sequence the first motion vector and minimum distortion are delivered on IOB bus of the STi3220 : they must be stored for future exploitation.

The FIFOs content at the end of the phase is:



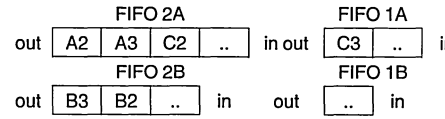
FIFO 1A has free location for writing the next search window block for the next process, i.e. A4. As this FIFO contains two blocks for search window, writing A4 can spend half an input block time (twice the pixel rate). In the same way B4 can be written into FIFO 1B during the time of one input block (at pixel rate).

- The 4th phase is again an initialisation sequence for the STi3220 during which the first part of the third sub search window must be loaded. During that phase blocks C1 and B1 are read out from FIFOs 2A and 2B. Both blocks will not be used anymore and hence are not written back into the FIFOs.

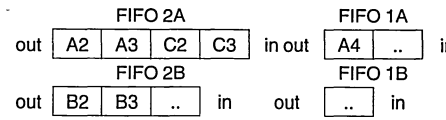


Note: The result of the second sub search window is available during that phase and must be stored.

- During the 5th phase, which is an STI3220's block sequence, blocks C2 and B2 are output from FIFOs 2A and 2B and must be written back into their original FIFO for future usage. Of course the same reference block is again input into the STI3220.



- The 6th is the last block sequence, pipelined with the previous one, during which the last part of the search window is loaded : block C3 is delivered by FIFO 1A and stored into 2A while block B3 is delivered by FIFO 2B.



At the end of that phase, FIFOs 2A and 2B are ready for starting the next reference block computation. The hypothesis of the FIFOs content at the beginning of the process is verified. For FIFOs 1A and 1B, we supposed at the beginning of the explanation that the blocks needed where already stored into the FIFOs. In fact blocks An and Bn are needed on the 3rd phase of the process, blocks Cn on the last phase. They can be loaded while they are not used and not necessarely since the beginning of a search window process.

Note : the result of the last sub search window is obtained after the 46th cycle of the first phase of the next search window delivery.

Note : The reference block must also be sent 4 times to the STi3220 chip. For that purpose the reference block is first sent to the board at 6f frequency and stored into a FIFO which output is sent back to its own input in the same way than the other FIFOs used for search window delivery.

V-5 Frame buffer accesses

The frame size is 352x288x1.5 = 152,064 bytes organised as 396 macro blocks each of 398 bytes. For serial port's use optimisation, each macro block is associated to one row of the memory. Only 396 rows over 512 and 398 columns over 512 are used in each of the two 256K x 4 video RAM of the frame buffer.

The serial port of the video RAM is dedicated to search window blocks reading and to block reconstruction : the blocks needed for search window are only Y component blocks (256 bytes) while the reconstructed block is a complete macro block (384 bytes).

When reconstructing a block into the frame memory, care must be taken in order not to scratch information still necessary for the search window (refer to example 1 for explanations): the delay before writing a new block into the frame memory after it has been computed in the STi3220 must be at least equal to 2 lines of blocks and 2 blocks = 46 macro blocks = 17664 cycles. This would be a lot of bytes to temporarely store in an additionnal memory. To avoid this, it is possible to write the reconstructed blocks directly into a free area of the frame memory (512 - 396 = 116 rows available) and to transfer after the desired delay, the macro blocks from the temporal additionnal area to their definitive location in the frame memory: this is one transfer from additionnal area in RAM to SAM and one transfer from SAM to frame area in RAM.

As the macro blocks are computed in the decoder in 8x8 size (in the order Y1 Y2 Y3 Y4 U and V as explained in chapter 4.1) it is necessary when writing the blocks into the frame memory to store them in the good order for a future macro block sequential output: first column of Y1 block must be followed by first column of Y3, followed by second column of Y1 .... That means that only 8 consecutive samples coming from the decoder can be written on consecutive addresses of the serial port. A transfer between SAM and RAM must be done every 8 pixels (except for U and V that are always 8x8 blocks stored after the last column of Y4). Writing a macro into the frame memory will need 32 transfers during Y samples block input and one final transfer after U and V blocks input.

The total number of accesses on the serial port during one block of pixels is:

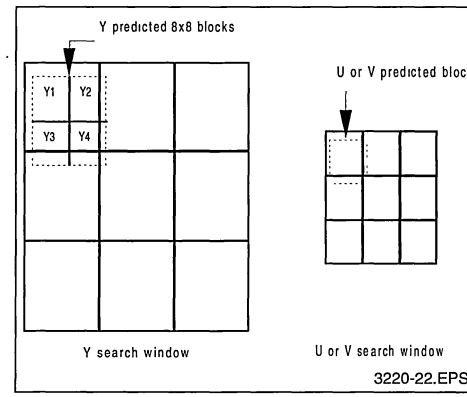
- read of 3 blocks of 256 bytes for search window. The maximum time for loading a block into serial memory consists in waiting for the current random access to be finished, followed by the transfer cycle from RAM to SAM. For 3 blocks this will be a total of 6 random cycles.
- write of the reconstructed macro block i.e. 384 serial cycles associated with 33 transfers and 33 "wait" cycles.
- 2 transfers between RAM and SAM for storage of the macro blocks in the good final address, associated to one "wait" cycle.

This is a total of 3x256 + 384 serial cycles + 6 + 66 + 3 random cycles. With a 100ns video RAM the minimum serial clock cycle is 30ns and the random one is 190ns leading to a global time of 48.810 us.

As the basic pixel frequency is 352 x 288 x 30Hz = 3.04128MHz = f, the total time allowed per block is 256/f = 84.175µs: the margin is high enough for the serial port accesses.

On the other hand, the random port is dedicated to the predicted macro block accesses.The predicted block must be delivered to the coder in a 8x8 form (Y1 Y2 Y3 Y4 U and V) (see Figure 23).

Figure 5.4 : Example of predictor position in the searc window



As it can be seen in the figure, for two 8x8 predicted blocks (Y1 and Y2 in the example above) it is possible to read 8 consecutive samples of a column on 8 consecutive addresses of a macro block and for two 8x8 predicted blocks (Y3 and Y4 in the example above) two different macro blocks access are necessary for delivering one column of 8 samples. In the same way for the U and V predicted blocks it is necessary to do two macro block changes every 8 samples. Each macro block change is a normal random cycle (190ns) while each consecutive address access can be done as a page mode cycle (55ns).

The total number of accesses for prediction is : 2 x (8 x (1 random + 7 page)) + 2 x (8 x (2 random + 6 page)) + 2 x (8 x (2 random + 6 page)) = 80 random + 304 page mode.

Those accesses will be interrupted by the control cycles for the serial port, i.e. 3 + 33 + 2 = 38 cycles (equivalent to a random access). As those ac-

cesses may happen during page mode sequences, we consider that each access for the serial port replaces a page mode cycle by a random cycle.

The global number of accesses on the random port is:

(80 + 38) random + 38 transfers + (304 - 38) page mode = 156 random + 266 page mode = 44.27µs.

This time is also compatible with the total time of 84.175µs allowed for a complete macro block input.

If we take a basic cycle time of 50ns for the serial port, with a page mode cycle of 100ns and a random cycle of 300ns, the total times on each port become:

- serial port : 1152 serial + 75 random = 80.1µs
- random port : 156 random + 266 page = 73.4µs

The time margin on the frame memory accesses may allow to use slower (and hence cheaper) video RAM, or free spaces in the vidoe RAM can be used for additional functions that have not been considered here, for example the line to block conversion for the reference block. The more functions will share the frame memory space the more complicated will be the address controller. However this controller is most likely to be realised with EPLD devices or integrated into an ASIC.

V-6 Conclusion

This architecture presents the advantage of offering interesting solutions for computing a search window range that didn't seem originally easy to manage:

- Use of only one STi3220 chip running full speed for calculation of the whole range.
- Use of a small amount of video RAM for frame buffer (two 256Kx4 chips) shared between frame reconstruction, prediction and serach window delivery.
- Use of 5 additional FIFOs connected in a clever way, for delivery of the good search window sequences on the STi3220 inputs.

However, having said that, it must be considered that the RAMs, FIFOs and multiplexer controller is going to be very complex. An ASIC circuit will be the good choice in that case.

VI - MISCELLANEOUS

VI-1 -16/+15 displacement range with one chip at lower speed

In the third example, we saw that the STi3220 chip was used at 6f frequency if f is the pixel rate, due to the fact that two intialisation sequences are necessary for each search window scanning. The pipeline mode cannot be used efficiently in this case.

However it is possible to always use the pipeline mode as shown in Figure 24.

All the blocks of a complete row are first compared to their associated sub-search window 1 for instance (upper left sub search window): all the corresponding partial vectors and distortions must be stored. Then the same reference blocks are sent again to the STi3220 chip for comparison with sub search window 2 (upper right): the resulting distortions are compared to their corresponding distortions of sub search window 1 and the minimum of both is kept in memory with its associated motion vector. Of course the same operation is made again with sub-search windows 3 (lower left) and 4 (lower right). During all the computation the pipeline mode is never broken.

In this case the chip's working frequency is only four times the pixel's rate.

After a complete line of blocks has been computed, all the resulting motion vectors can be used for extraction of the predicted blocks.

The disadvantage of this solution is that it is necessary to work over a complete line of blocks (address generation more complicated) and that, consequently it needs the storage of all the minimum distortions and motion vectors of a line of block.

This solution could be useful for applications where the STi3220 working frequency is not high enough (chip running at 4f instead of 6f).

VI-2 Computing larger displacement ranges

There is no limitation on the size of the search window that can be used : the only condition is to split the global search window into sub-search windows compatible with the STi3220 chip search window (-8 to +7). Figure 25 shows an example of -32 to + 31 search window scanning in horizontal direction and -16 to +15 scanning in vertical direction with 16x16 blocks.

The larger the total search window, the higher the number of sub-search windows and the faster the chip's working frequency. In the example of Figure 25 the chip must work at 10 times the pixel's frequency. If the chip frequency becomes too high it is then necessary to use several chips as explained in chapter VI.3.

Figure 24 : -16/+15 Displacement Range with STI3220 at 4f Frequency

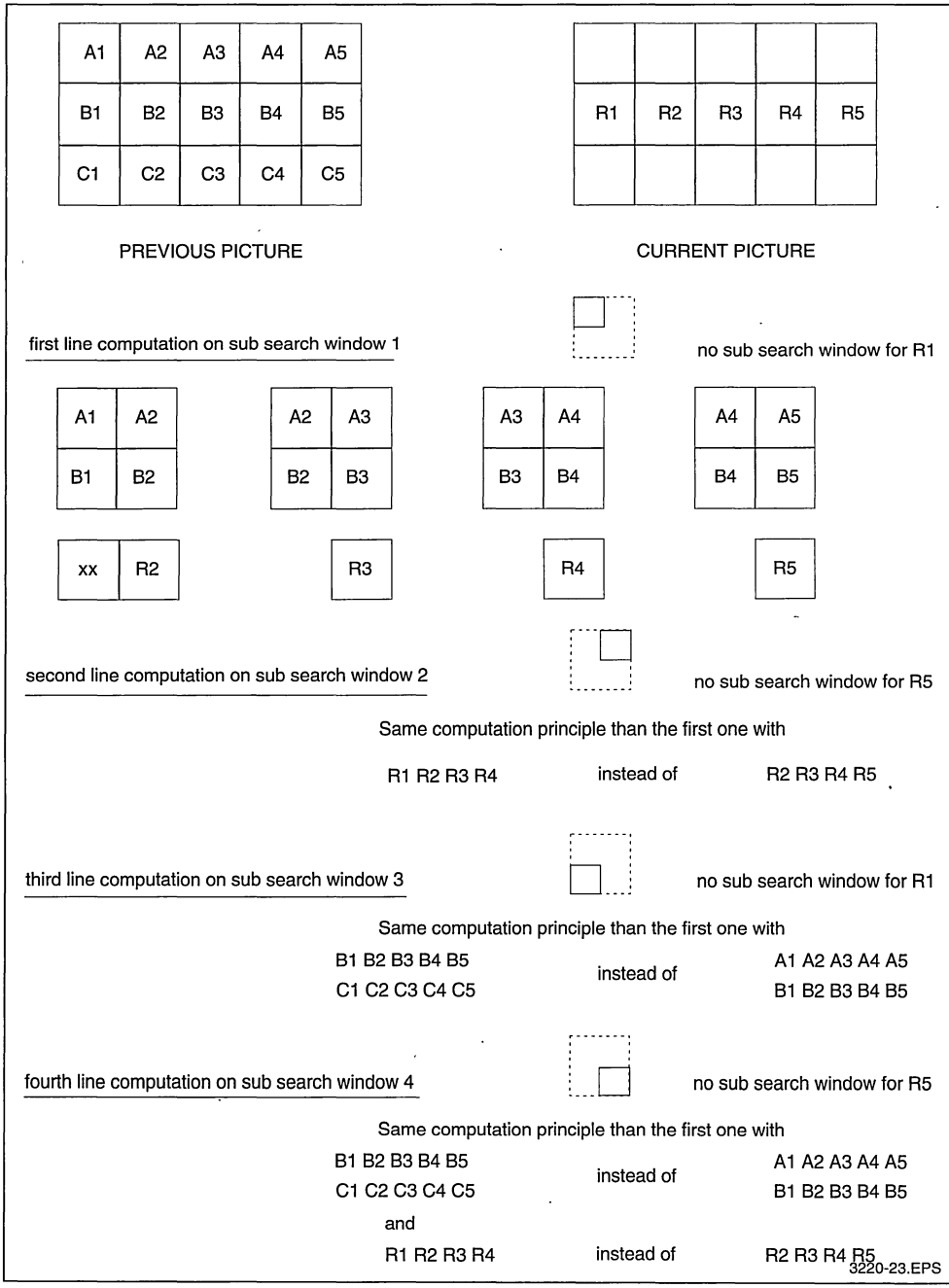


Figure 25 : Example of Search Window Range -32/+31 in Horizontal Direction -16/+15 In Vertical Direction

A1	A2	A3	A4	A5
B1	B2	B3	B4	B5
C1	C2	C3	C4	C5

search window of reference block centered on B3 position

sub search window 1 ·	A1 A2	sub search window 5	B1 B2
1 initialisation + 1 block sequence	B1 B2	1 initialisation + 1 block sequence	C1 C2
sub search window 2	a2 A3	sub search window 6	b2 B3
1 block sequence	b2 B3	1 block sequence	c2 C3
sub search window 3	a3 A4	sub search window 7	b3 B4
1 block sequence	b3 B4	1 block sequence	c3 C4
sub search window 4	a4 A5	sub search window 8	b4 B5
1 block sequence	b4 B5	1 block sequence	c4 C5

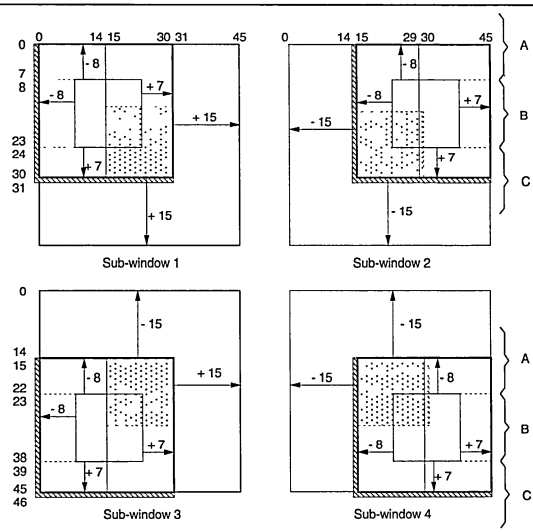
An, Bn or Cn : blocks sent to the STi 3220

an, bn, cn : blocks already in the STi 3220

Total time for scanning all the search window = 10 blocks

3220-24.EPS

Figure 26 : -15/+15 Search Window



3220-25.EPS

The search window size is not necessarely defined between -(m x block_size) to (m x block_size)- 1 as all the examples that have been seen before. But if it is not, then the pipeline mode cannot be used efficiently on a line of blocks. The main typical case concerns the 16x16 blocks with -15 to +15 search window range. As it can be seen on Figure 26, the initialisation sequence of sub-search window 2 (or 4) is not equal to the blocks sequence of sub-search window 1 (or 3) but is shifted one column left: therefore the pipeline mode cannot be used between search windows 1 (3) and 2 (4).

The time needed to scan the complete search window is equivalent to 8 sequences (4 initialisations and 4 blocks) instead of the 6 sequences (2 intialisations and 4 blocks).

If the STi3220 motion estimation chip is not able to support the pixel rate implied by such an application, an alternative could be to compute the -16/+15 search window and to clamp the motion vectors that may be equal to -16 to the value -15: this is not critical as in that case the prediction is always made but perhaps with less efficiency.

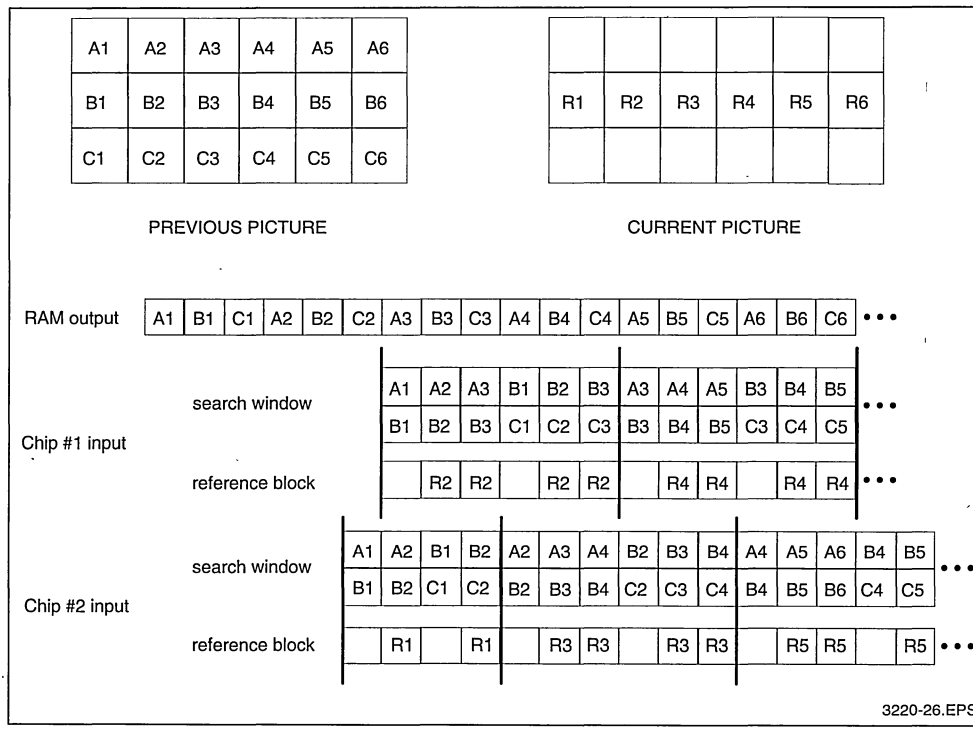
VI-3 Using several STi3220

If the STi3220 working frequency is not high enough for the desired search window and pixels' frequency, it becomes necessary to use several chips for the computations. Two solutions are then possible:

- Each chip computes one or several sub search windows of each reference block.
- Each chip computes the complete search window but only for one reference block every N reference blocks (where N is the number of chips) as illustrated in the figure 6.4 here under with two STi3220.

In that example, it can be noticed that the frame buffer RAM working frequency is exactly the same as the STi3220 chips working frequency. Of course a small additional RAM is necessary around each chip for temporal storage of its search window. The motion vector is alternatively delivered by one chip or the other.

Figure 27 : Use of Two STi3220



EUROPE

DENMARK

2730 HERLEV
Herlev Torv, 4
Tel. (45-42) 94.85.33
Telex: 35411
Telefax: (45-42) 948694

FINLAND

LOHJA SF-08150
Karjalankatu, 2
Tel. (358-12) 155.11
Telefax. (358-12) 155 66

FRANCE

94253 GENTILLY Cedex
7 - avenue Gallieni - BP. 93
Tel.. (33-1) 47.40.75.75
Telex: 632570 STMHQ
Telefax: (33-1) 47.40.79.10

67000 STRASBOURG
20, Place des Halles
Tel. (33) 88 75.50 66
Telefax: (33) 88.22.29.32

GERMANY

8011 GRASBRUNN
Bretonischer Ring 4
Postfach 1122
Tel.: (49-89) 460060
Telefax: (49-89) 4605454
Teletex: 897107=STDISTR

1000 BERLIN 37
Clay Allee 323
Tel.: (49-30) 8017087-89
Telefax: (49-30) 8015552

6000 FRANKFURT
Gutleutstrasse 322
Tel. (49-69) 237492-3
Telefax: (49-69) 231957
Teletex: 6997689=STVBF

3000 HANNOVER 51
Rotenburger Strasse 28A
Tel. (49-511) 615960
Teletex. 5118418 CSFBEH
Telefax. (49-511) 6151243

5202 HENNEF
Reuther Strasse 1A-C
Tel. (49-2242) 6088
(49-2242) 4019/4010
Telefax· (49-2242) 84181

8500 NÜRNBERG 20
Erlenstegenstrasse, 72
Tel.: (49-911) 59893-0
Telefax: (49-911) 5980701

7000 STUTTGART 31
Mittlerer Pfad 2-4
Tel. (49-711) 13968-0
Telefax: (49-711) 8661427

ITALY

20090 ASSAGO (MI)
V.le Milanofiori - Strada 4 - Palazzo A/4/A
Tel. (39-2) 89213.1 (10 linee)
Telex: 330131 - 330141 SGSAGR
Telefax: (39-2) 8250449

40033 CASALECCHIO DI RENO (BO)
Via R. Fucini, 12
Tel. (39-51) 591914
Telex: 512442
Telefax: (39-51) 591305

00161 ROMA
Via A. Torlonia, 15
Tel. (39-6) 8443341
Telex: 620653 SGSATE I
Telefax: (39-6) 8444474

NETHERLANDS

5652 AR EINDHOVEN
Meerenakkerweg 1
Tel.: (31-40) 550015
Telex: 51186
Telefax: (31-40) 528835

SPAIN

08021 BARCELONA
Calle Platon, 6 4th Floor, 5th Door
Tel. (34-3) 4143300-4143361
Telefax: (34-3) 2021461

28027 MADRID
Calle Albacete, 5
Tel (34-1) 4051615
Telex: 46033 TCCEE
Telefax: (34-1) 4031134

SWEDEN

S-16421 KISTA
Borgarfjordsgatan, 13 - Box 1094
Tel.: (46-8) 7939220
Telex: 12078 THSWS
Telefax: (46-8) 7504950

SWITZERLAND

1218 GRAND-SACONNEX (GENEVA)
Chemin Francois-Lehmann, 18/A
Tel. (41-22) 7986462
Telex: 415493 STM CH
Telefax: (41-22) 7984869

UNITED KINGDOM and EIRE

MARLOW, BUCKS
Planar House, Parkway
Globe Park
Tel.: (44-628) 890800
Telex: 847458
Telefax: (44-628) 890391

AMERICAS

BRAZIL

05413 SÃO PAULO
R. Henrique Schaumann 286-CJ33
Tel. (55-11) 883-5455
Telex: (391)11-37988 "UMBR BR"
Telefax : (55-11) 282-2367

U.S.A.

NORTH & SOUTH AMERICAN
MARKETING HEADQUARTERS
1000 East Bell Road
Phoenix, AZ 85022
(1-602) 867-6100

SALES COVERAGE BY STATE

ALABAMA
Huntsville - (205) 533-5995

ARIZONA
Phoenix - (602) 867-6217

CALIFORNIA
Santa Ana - (714) 957-6018
San Jose - (408) 452-8585

COLORADO
Boulder (303) 449-9000

ILLINOIS
Schaumburg - (708) 517-1890

INDIANA
Kokomo - (317) 455-3500

MASSACHUSETTS
Lincoln - (617) 259-0300

MICHIGAN
Livonia - (313) 953-1700

NEW JERSEY
Voorhees - (609) 772-6222

NEW YORK
Poughkeepsie - (914) 454-8813

NORTH CAROLINA
Raleigh - (919) 787-6555

TEXAS
Carrollton - (214) 466-8844

FOR RF AND MICROWAVE
POWER TRANSISTORS CON-
TACT
THE FOLLOWING REGIONAL
OFFICE IN THE U.S.A.

PENNSYLVANIA
Montgomeryville - (215) 361-6400

ASIA / PACIFIC

AUSTRALIA

NSW 2220 HURTSVILLE
Suite 3, Level 7, Otis House
43 Bridge Street
Tel. (61-2) 5803811
Telefax: (61-2) 5806440

HONG KONG

WANCHAI
22nd Floor - Hopewell centre
183 Queen's Road East
Tel. (852) 8615788
Telex: 60955 ESGIES HX
Telefax· (852) 8656589

INDIA

NEW DELHI 110001
LiasonOffice
62, Upper Ground Floor
World Trade Centre
Barakhamba Lane
Tel. (91-11) 3715191
Telex: 031-66816 STMI IN
Telefax. (91-11) 3715192

MALAYSIA

PULAU PINANG 10400
4th Floor - Suite 4-03
Bangunan FOP-123D Jalan Anson
Tel (04) 379735
Telefax (04) 379816

KOREA

SEOUL 121
8th floor Shinwon Building
823-14, Yuksam-Dong
Kang-Nam-Gu
Tel. (82-2) 553-0399
Telex SGSKOR K29998
Telefax (82-2) 552-1051

SINGAPORE

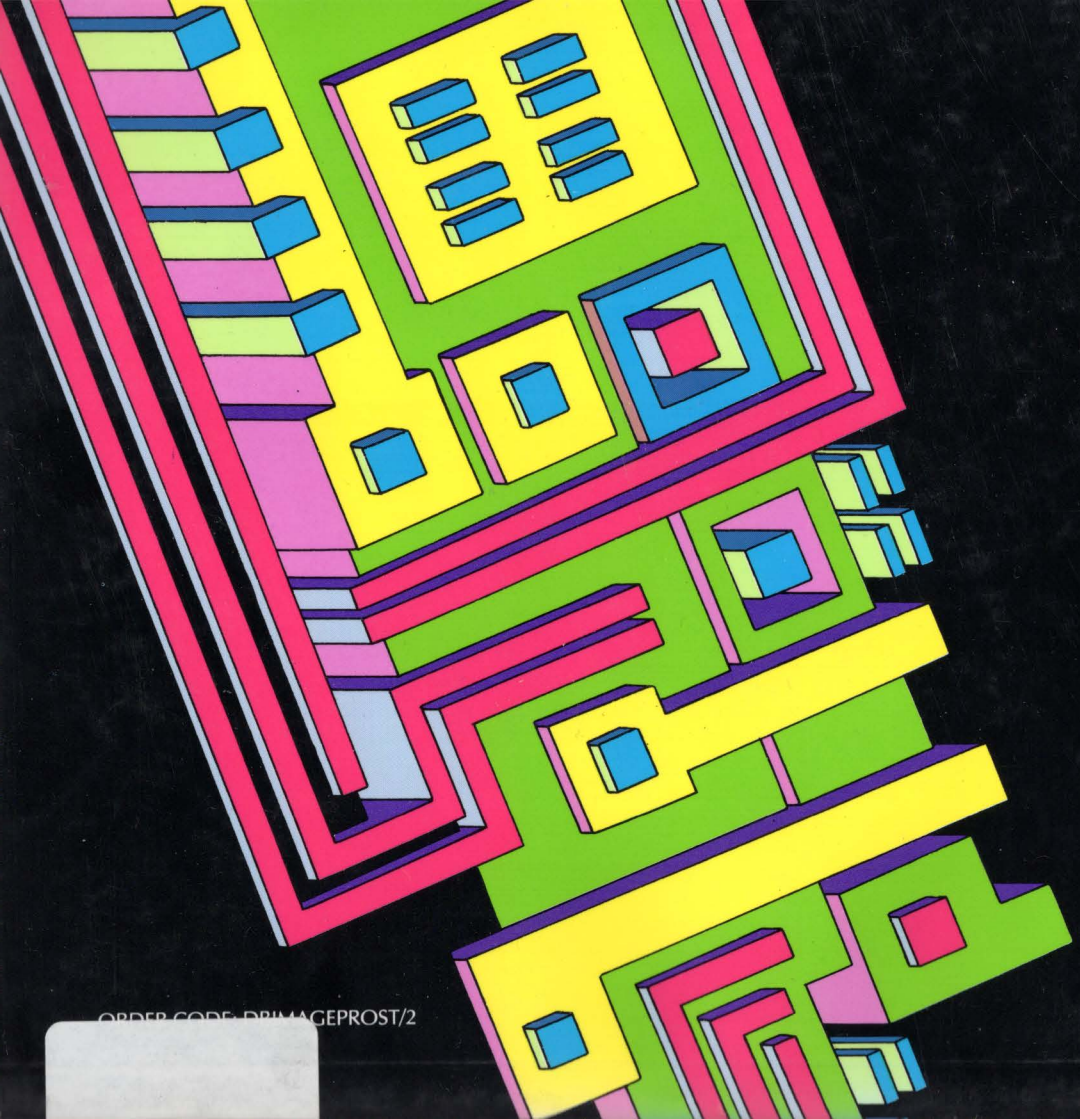
SINGAPORE 2056
28 Ang Mo Kio - Industrial Park 2
Tel (65) 4821411
Telex· RS 55201 ESGIES
Telefax. (65) 4820240

TAIWAN

TAIPEI
12th Floor
325, Section 1 Tun Hua South Road
Tel (886-2) 755-4111
Telex. 10310 ESGIE TW
Telefax (886-2) 755-4008

JAPAN

TOKYO 108
Nisseki - Takanawa Bld. 4F
2-18-10 Takanawa
Minato-Ku
Tel. (81-3) 3280-4121
Telefax: (81-3) 3280-4131



ORDER CODE: PRIMAGEPROST/2